Fakultät für
Informatik und
Mathematik

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

COMPUTER ARCHITECTURE

Chapter 4 – Complex Pipelining

Prof. Dr.-Ing. Stefan Wallentowitz

Department 07 – Munich University of Applied Sciences

This work is licensed under a Creative Commons Attribution 4.0 International License.

Course Organization

GPU

[rwindr, Pixabay]

PCIe

Peripherals

[MichaelOpdenacker, CC BY-SA 4.0]

SATA

[WallpaperSafari, Public Domain]

FSB

[Intel DE, CC BY-SA 2.0]

South Bridge

North Bridge

Central Processing Unit (CPU)

[George Hodan, CC0]

USB

DDR

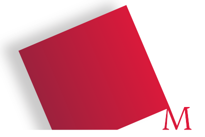
Caches

- Basics
- Pipelining
- Compley Pipelining

[Martyx, CC BY-SA 3.0]

Main Memory

Course Organization



GPU

PCIe

Peripherals

SATA

USB

FSB

South Bridge

North Bridge

DDR

Main Memory

[rwindr, Pixabay]

[MichaelOpdenacker, CC BY-SA 4.0]

[WallpaperSafari, Public Domain]

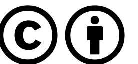
[George Hodan, CC0]

[Intel DE, CC BY-SA 2.0]

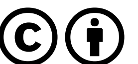
[Martyx, CC BY-SA 3.0]

Central Processing Unit (CPU)

Caches
- Basics
- Pipelining
- Compley Pipelining



Pipeline Optimizations



Pipeline Optimizations

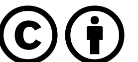
Goal: Bring IPC up (near to one or even above)

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

Prof. Dr.-Ing. Stefan Wallentowitz – Department 07 – Munich University of Applied Sciences

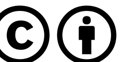


Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions



Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

- Instruction Level Parallelism (ILP)

CC BY

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

- Instruction Level Parallelism (ILP)
  - Pipelining

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

- Instruction Level Parallelism (ILP)
  - Pipelining
  - Superscalar execution, out-of-order execution (lecture part 4)

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

- Instruction Level Parallelism (ILP)
  - Pipelining
  - Superscalar execution, out-of-order execution (lecture part 4)
- Data parallelism

ⓒ ⓘ

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

- Instruction Level Parallelism (ILP)
  - Pipelining
  - Superscalar execution, out-of-order execution (lecture part 4)
- Data parallelism
  - Data vectors, single instruction multiple data

CC BY

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

- Instruction Level Parallelism (ILP)
  - Pipelining
  - Superscalar execution, out-of-order execution (lecture part 4)
- Data parallelism
  - Data vectors, single instruction multiple data
- Thread parallelism

Pipeline Optimizations

Goal: Bring IPC up (near to one or even above)

Speculative Execution

- Branch prediction: Reduce the impact of branch decisions
- Other kinds of speculation: Address, data, ...

Parallelism

- Instruction Level Parallelism (ILP)
  - Pipelining
  - Superscalar execution, out-of-order execution (lecture part 4)
- Data parallelism
  - Data vectors, single instruction multiple data
- Thread parallelism
  - Execution of multiple different instruction streams

Recap: Pipelining

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

Prof. Dr.-Ing. Stefan Wallentowitz – Department 07 – Munich University of Applied Sciences

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

- General exception:Memory accesses take multiple cycles

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

- General exception:Memory accesses take multiple cycles

Implementation of execute stage:

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

- General exception:Memory accesses take multiple cycles

Implementation of execute stage:

- Basically: Arithmetic and Logical Unit (ALU)

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

- General exception:Memory accesses take multiple cycles

Implementation of execute stage:

- Basically: Arithmetic and Logical Unit (ALU)
- But also:

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

- General exception:Memory accesses take multiple cycles

Implementation of execute stage:

- Basically: Arithmetic and Logical Unit (ALU)
- But also:
  - Branch offset ALU

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

- General exception:Memory accesses take multiple cycles

Implementation of execute stage:

- Basically: Arithmetic and Logical Unit (ALU)
- But also:
  - Branch offset ALU
  - Multiplier/Divider (RISC-V M extension)

Recap: Pipelining

Assumption: Each instruction takes one cycle per stage

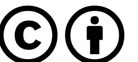
- General exception:Memory accesses take multiple cycles

Implementation of execute stage:

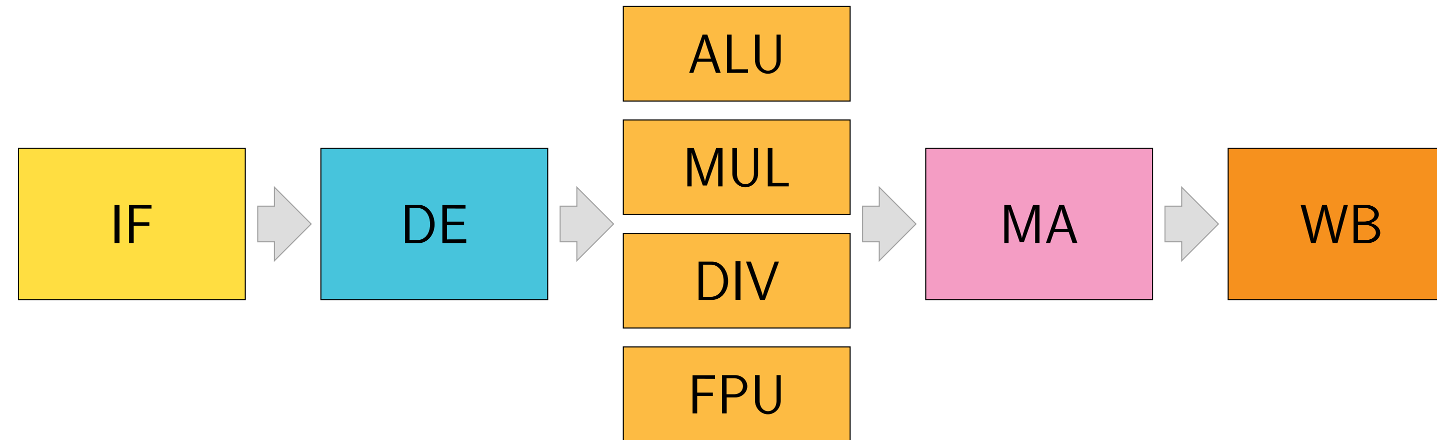
- Basically: Arithmetic and Logical Unit (ALU)
- But also:
  - Branch offset ALU
  - Multiplier/Divider (RISC-V M extension)
  - Floating Point Unit (RISC-V F/D extension)

(cc) (by)

Pipeline: Functional Units

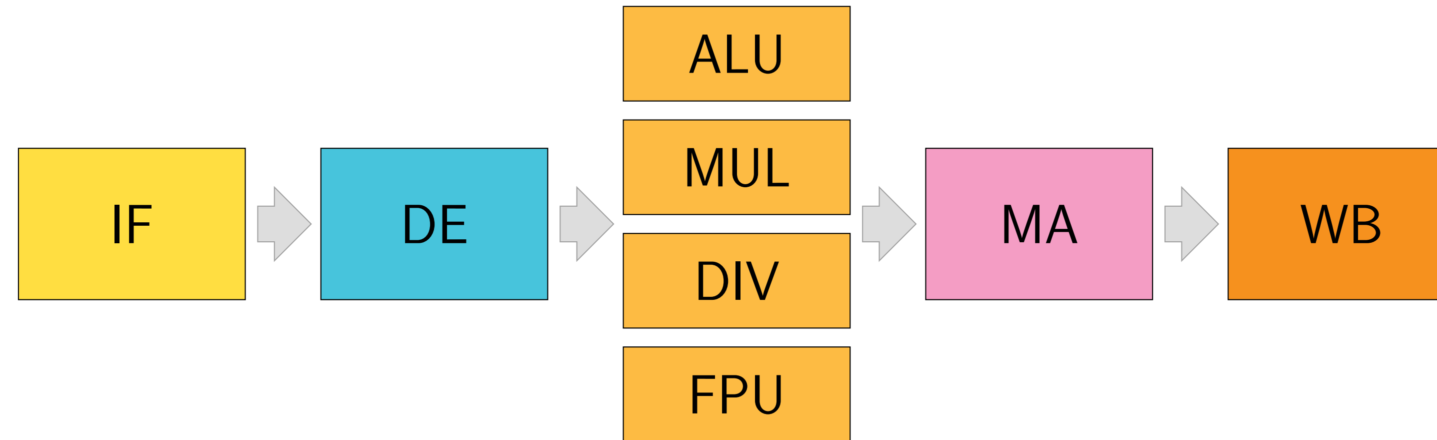


Pipeline: Functional Units



Split EX into functional units (FU): Different hardware building blocks

Pipeline: Functional Units

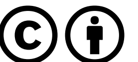


Split EX into functional units (FU): Different hardware building blocks

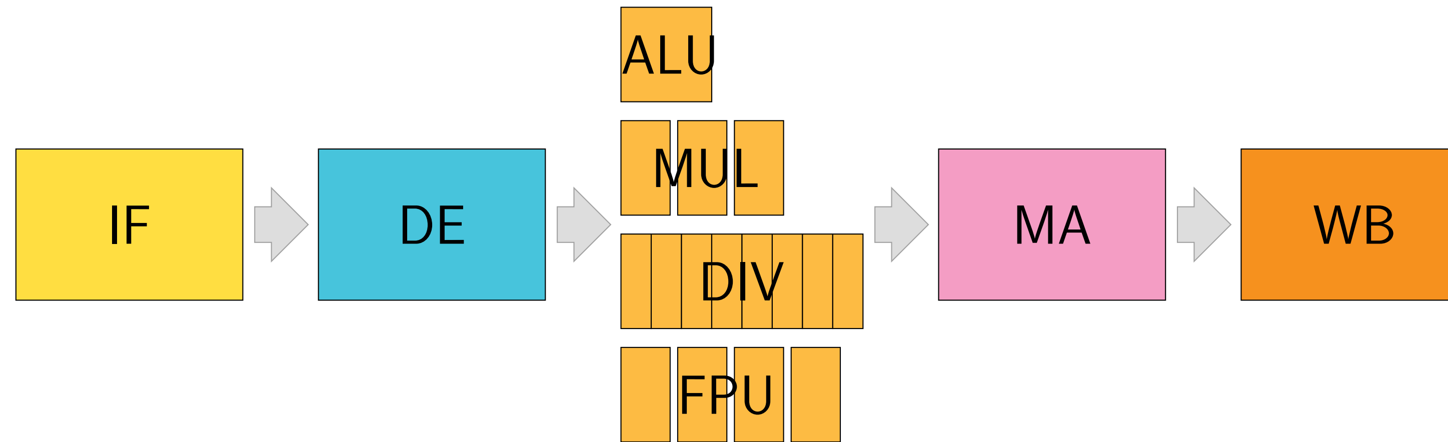
Multicycle instructions: Instructions don't complete in one cycle:
Multiplier, Divider, Floating Point Unit (FPU)

(cc) (by)

Functional Units: Multicycle Pipelining

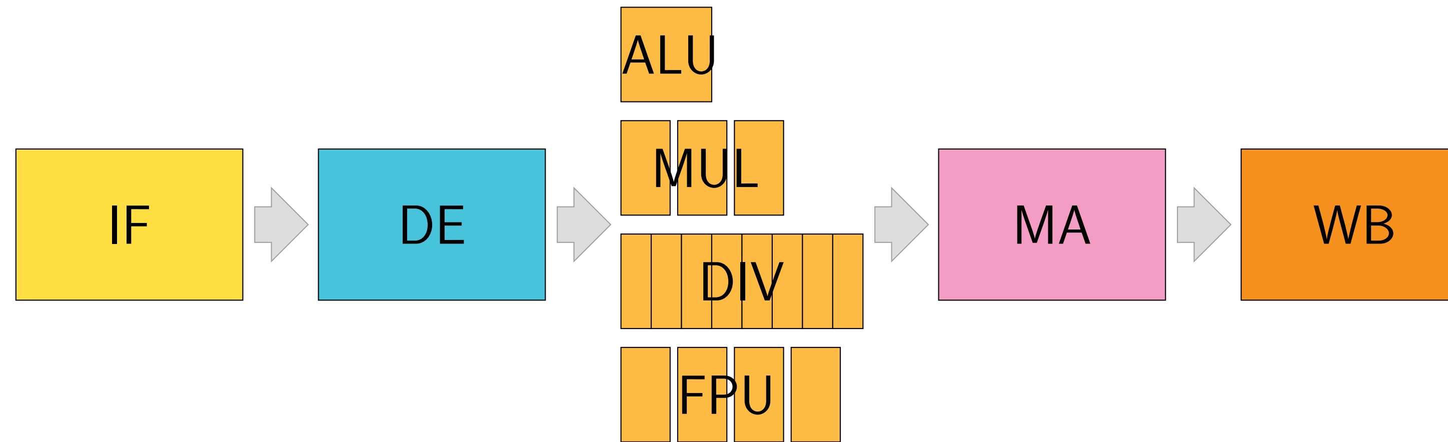


Functional Units: Multicycle Pipelining



Multicycle FUs may be pipelined: Decomposition of operation

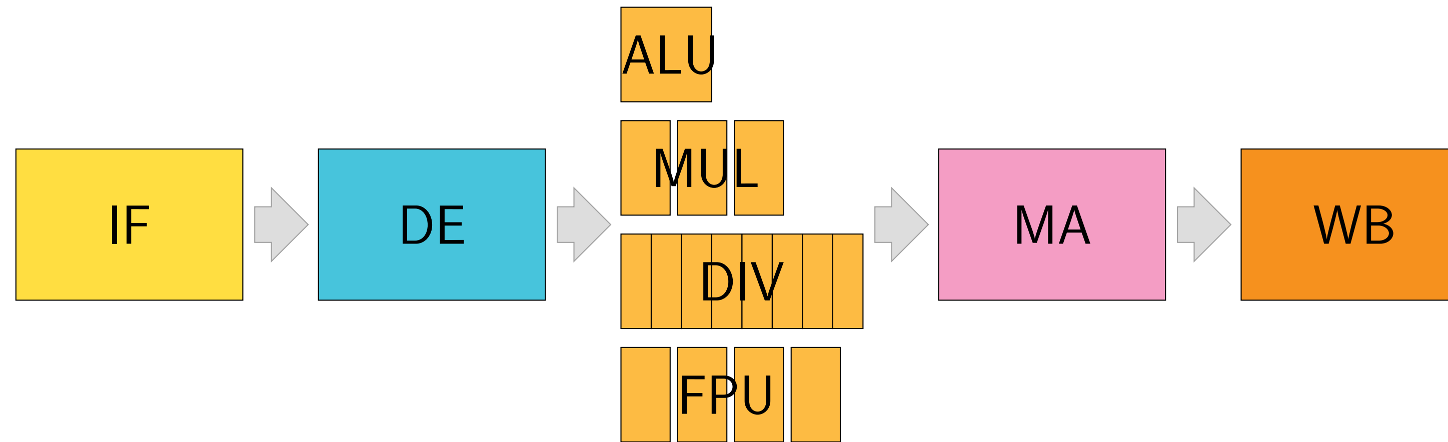
Functional Units: Multicycle Pipelining



Multicycle FUs may be pipelined: Decomposition of operation

Sometimes not possible: DIV often shares one unit over multiple cycles

Functional Units: Multicycle Pipelining

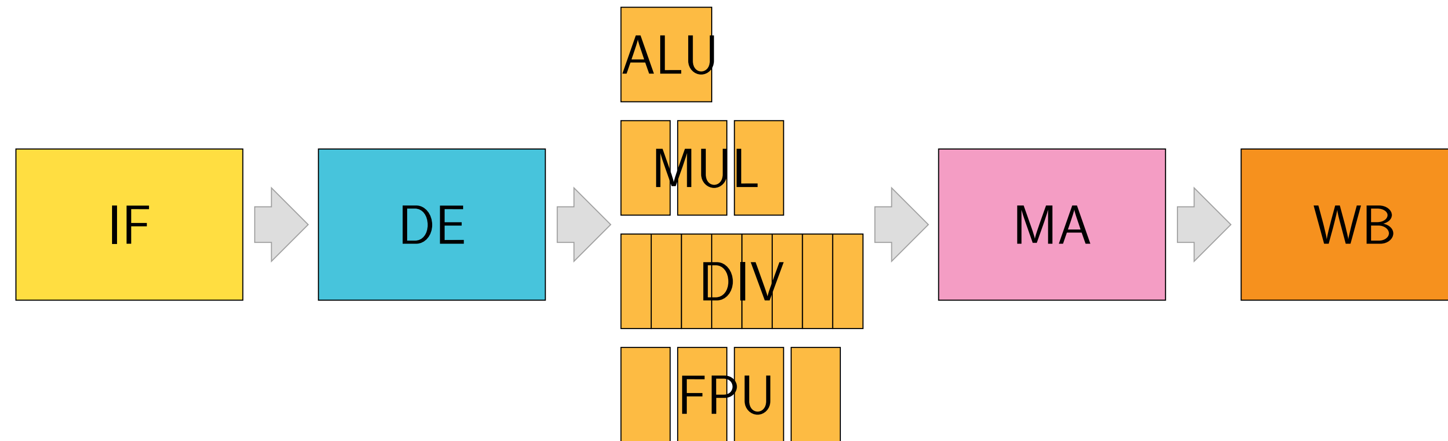


Multicycle FUs may be pipelined: Decomposition of operation

Sometimes not possible: DIV often shares one unit over multiple cycles

Allows for parallel execution of multiple instructions in one FU (not always the case)

Functional Units: Multicycle Pipelining



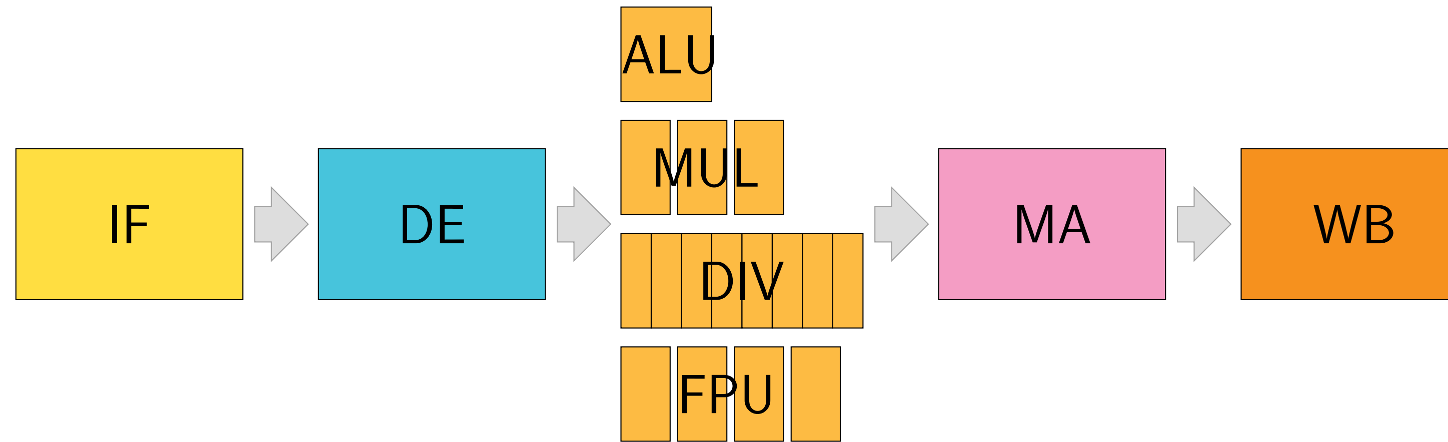
Multicycle FUs may be pipelined: Decomposition of operation

Sometimes not possible: DIV often shares one unit over multiple cycles

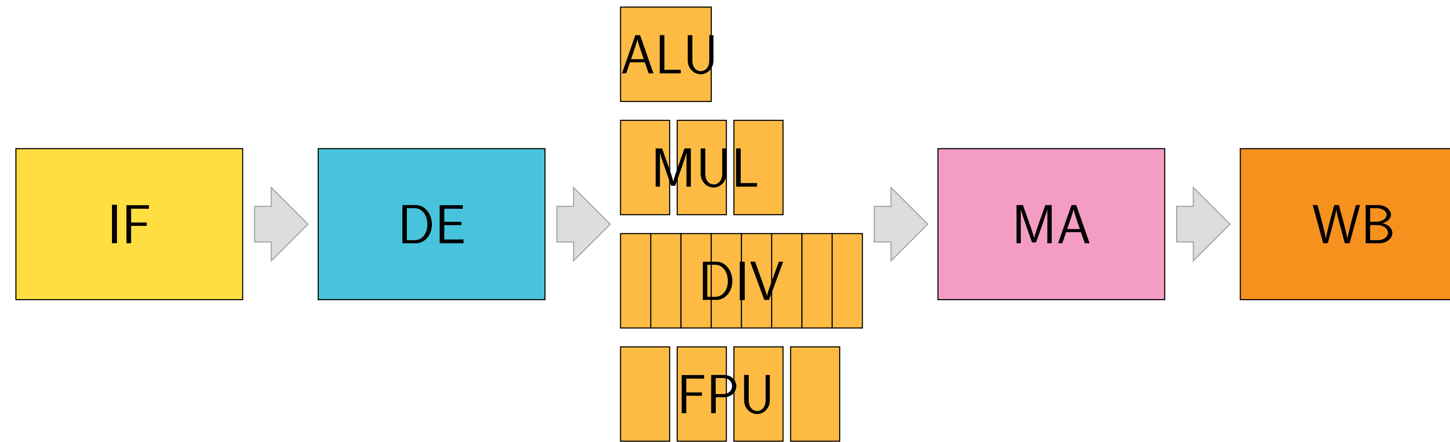
Allows for parallel execution of multiple instructions in one FU (not always the case)

(note: In the diagram each block corresponds to one clock cycle, differently scaled)

Multicycle Metrics

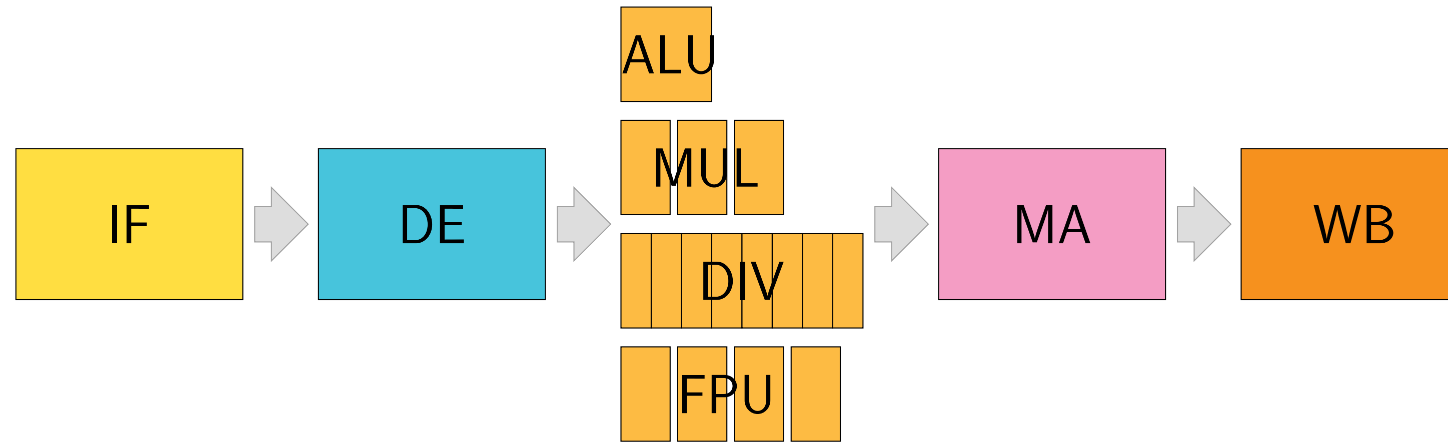


Multicycle Metrics



Latency

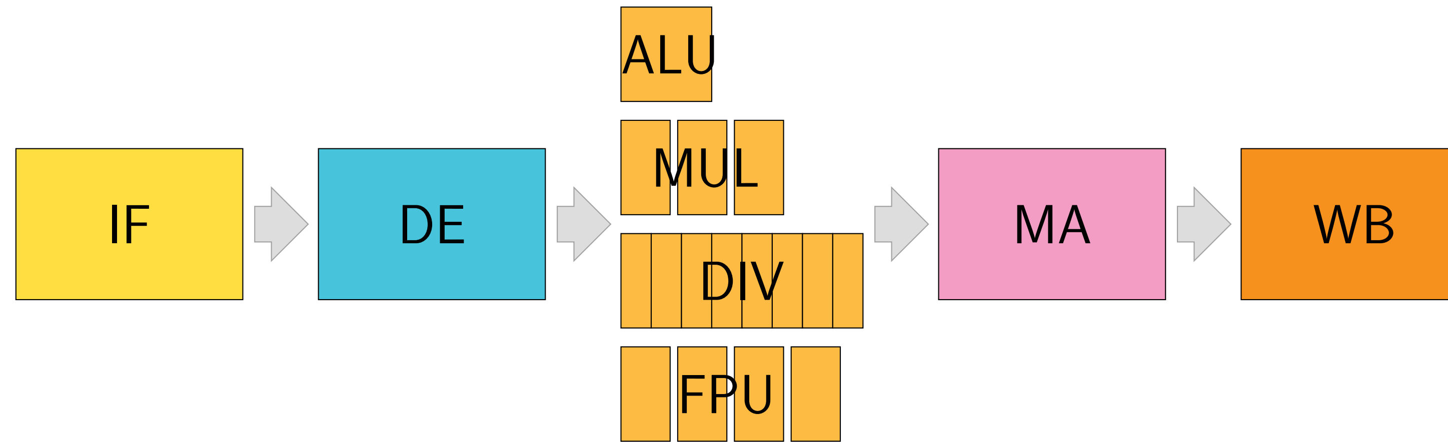
Multicycle Metrics



Latency

- Minimal time for instruction to traverse a functional unit

Multicycle Metrics



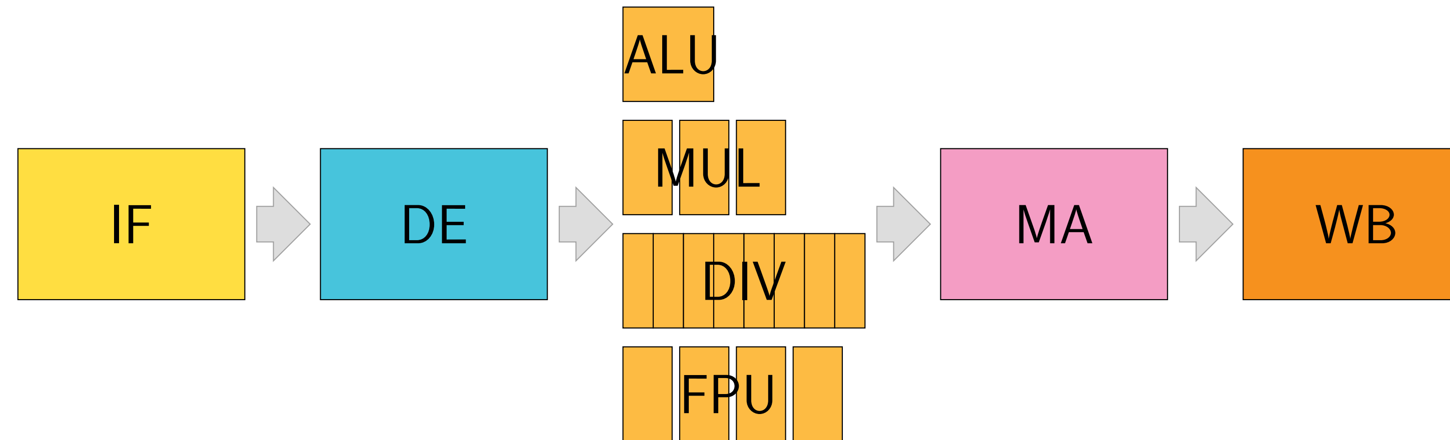
Latency

- Minimal time for instruction to traverse a functional unit

Initiation Interval

© ⓘ

Multicycle Metrics



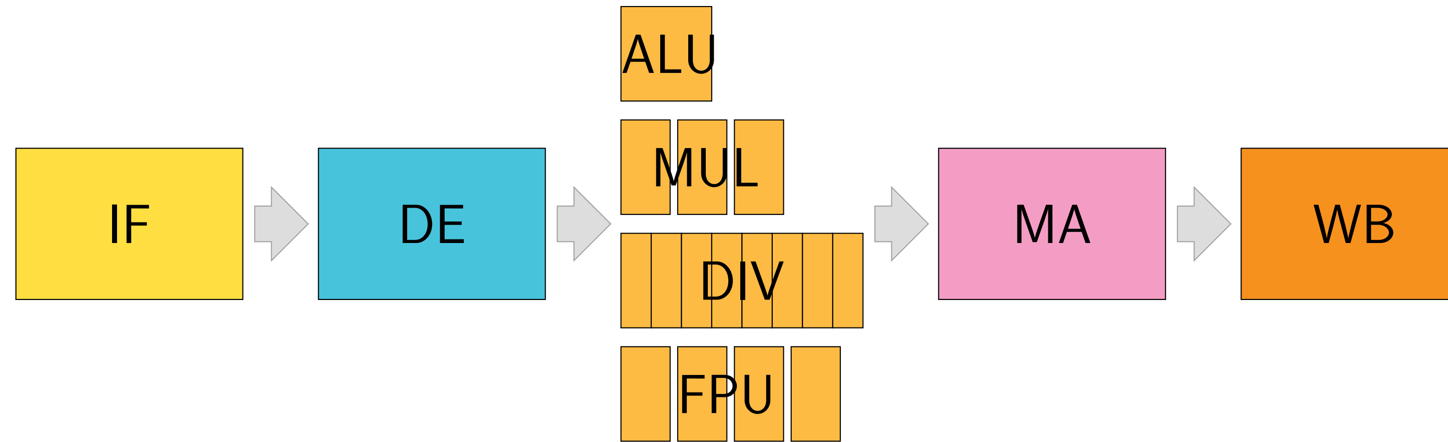
Latency

- Minimal time for instruction to traverse a functional unit

Initiation Interval

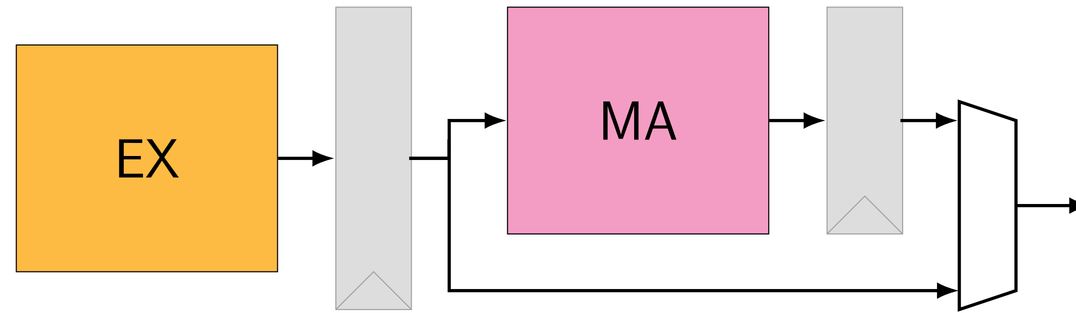
- Minimal duration between two instructions can be started in a functional unit

Multicycle Metrics



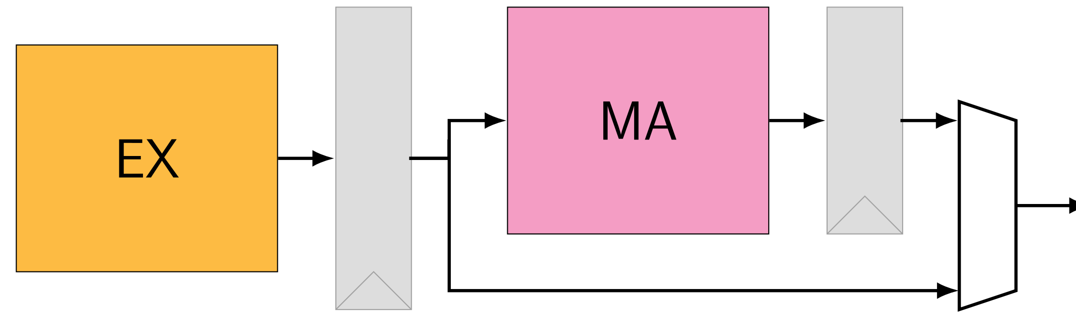
	Latency	Initiation Interval
(Integer) ALU		
Multiplier		
Divider		
FPU		

Load-Store Access



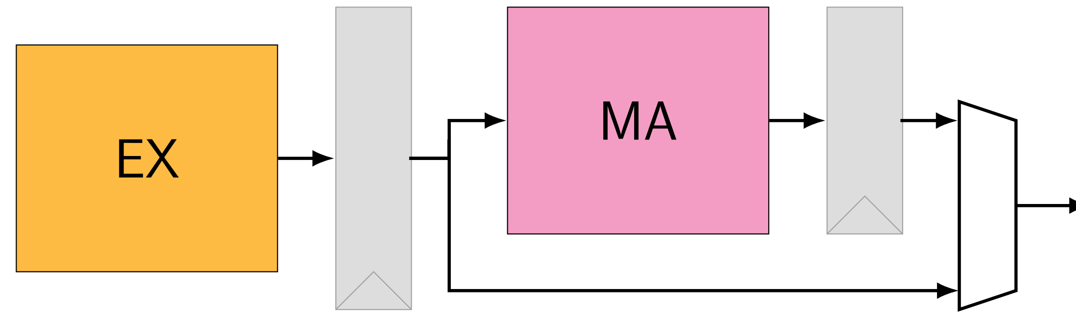
Prof. Dr.-Ing. Stefan Wallentowitz – Department 07 – Munich University of Applied Sciences

Load-Store Access



Memory access can be optional as most operations don't use it

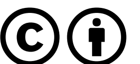
Load-Store Access



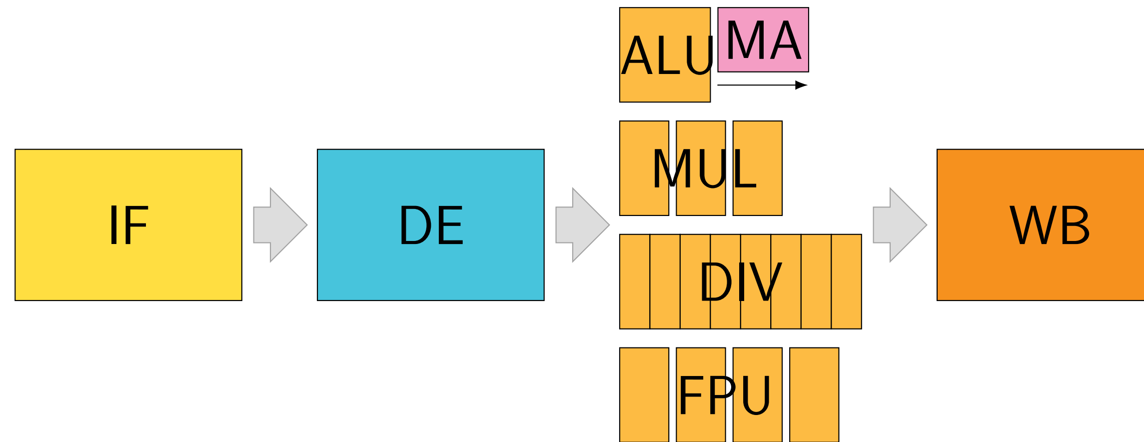
Memory access can be optional as most operations don't use it

Specifically after split into functional units

Load-Store Access

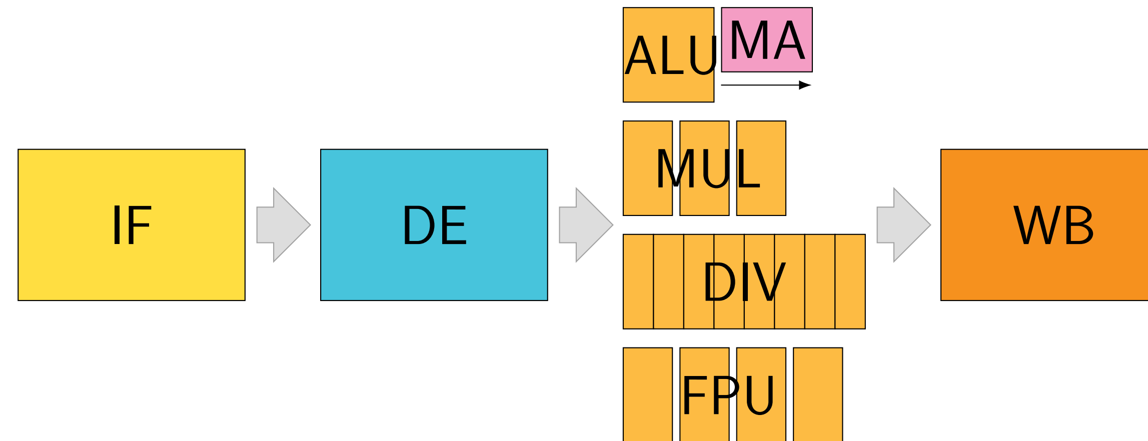


Load-Store Access



In pipeline with FUs, MA can be an optional extra stage after ALU

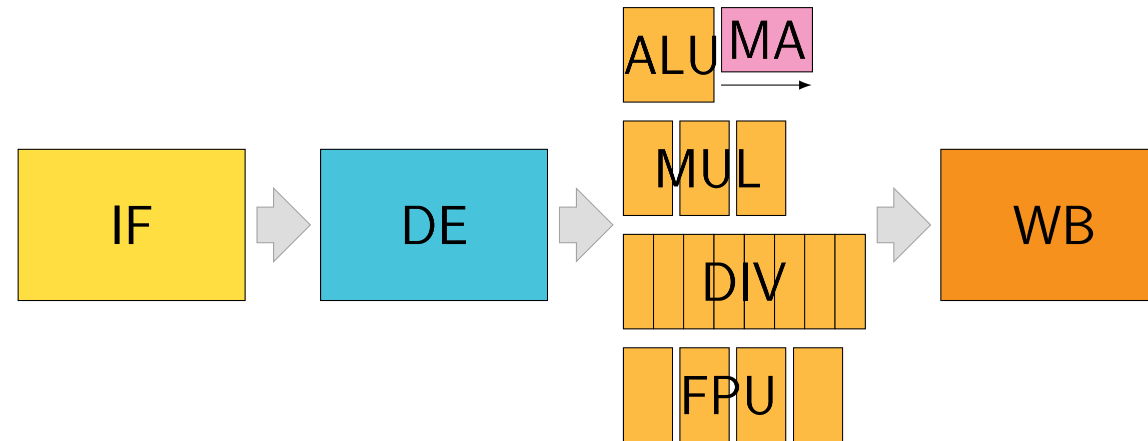
Load-Store Access



In pipeline with FUs, MA can be an optional extra stage after ALU

- Other paths are not concerned

Load-Store Access



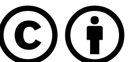
In pipeline with FUs, MA can be an optional extra stage after ALU

- Other paths are not concerned
- For ALU operations its optional to traverse MA

Example: Only one instruction in FUs

Example: Only one instruction in FUs

Similar as before: Only one instruction can be in any FU at any time



Example: Only one instruction in FUs

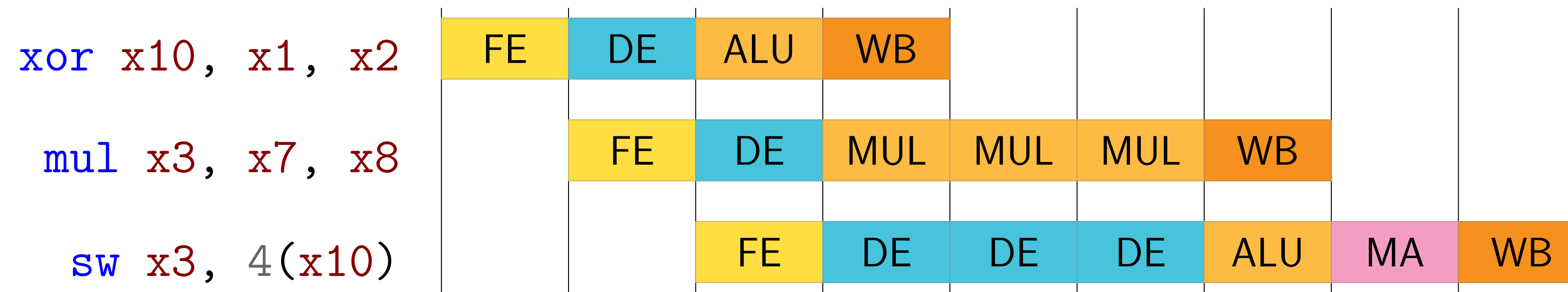
Similar as before: Only one instruction can be in any FU at any time

Structural hazard for multicycle operations

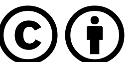
Example: Only one instruction in FUs

Similar as before: Only one instruction can be in any FU at any time

Structural hazard for multicycle operations

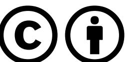


Example: Overlap FU processing



Example: Overlap FU processing

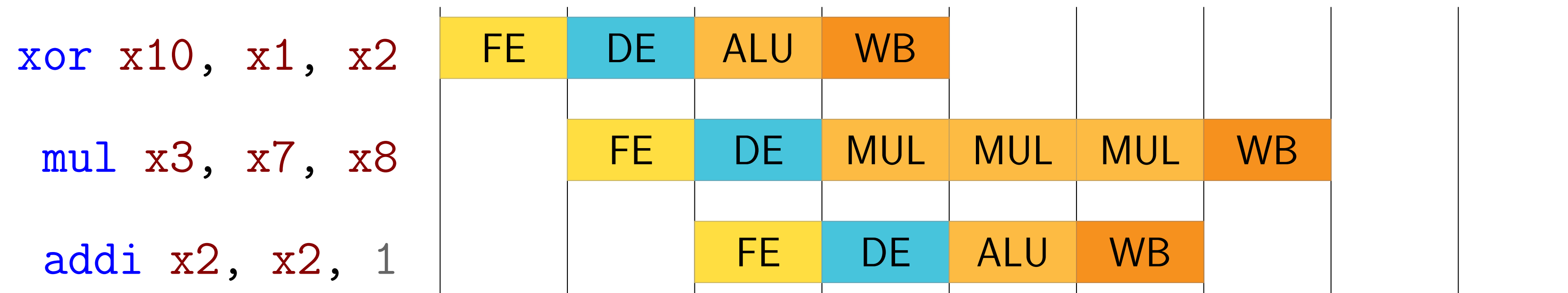
Issue at most one instruction per cycle



Example: Overlap FU processing

Issue at most one instruction per cycle

But: multicycle instructions may still be ongoing

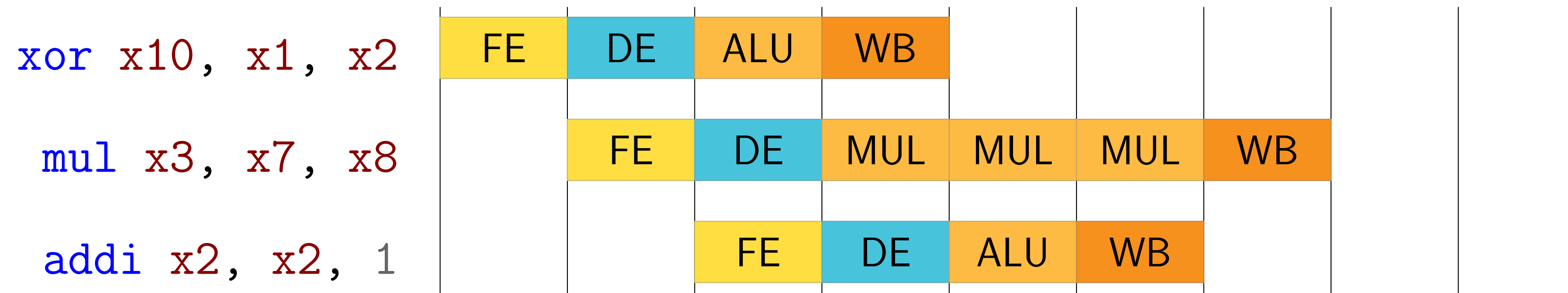


(c)(i)

Example: Overlap FU processing

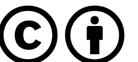
Issue at most one instruction per cycle

But: multicycle instructions may still be ongoing



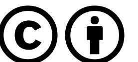
Due to different latency: Instructions can "overtake" others

Out-of-Order Completion



Out-of-Order Completion

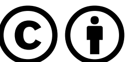
Even when started in correct order, instructions can complete *out-of-order*



Out-of-Order Completion

Even when started in correct order, instructions can complete *out-of-order*

Structural hazard on writeback stage, can be resolved



Out-of-Order Completion

Even when started in correct order, instructions can complete *out-of-order*

Structural hazard on writeback stage, can be resolved

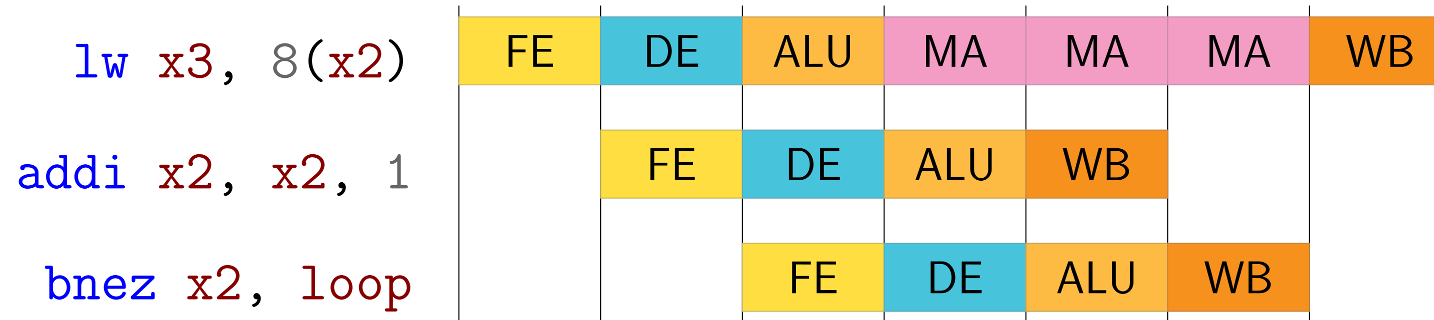
Example:

Out-of-Order Completion

Even when started in correct order, instructions can complete *out-of-order*

Structural hazard on writeback stage, can be resolved

Example:

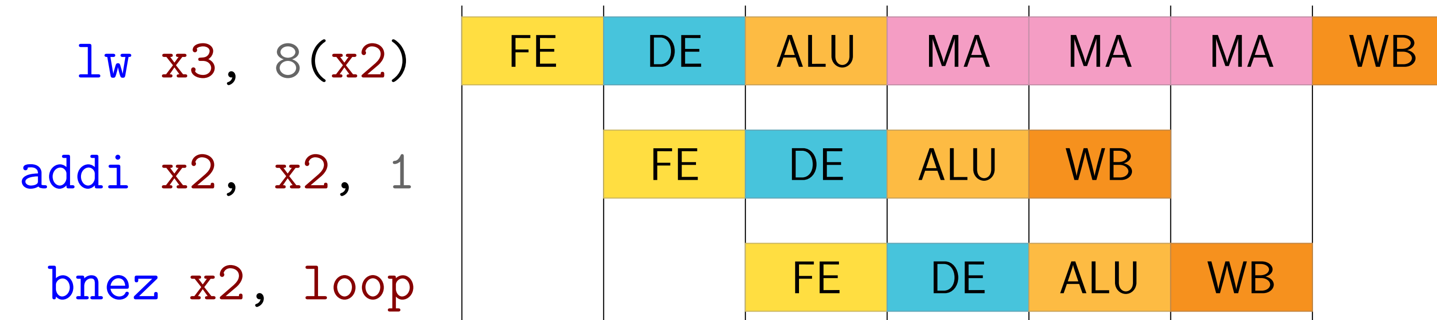


Out-of-Order Completion

Even when started in correct order, instructions can complete *out-of-order*

Structural hazard on writeback stage, can be resolved

Example:



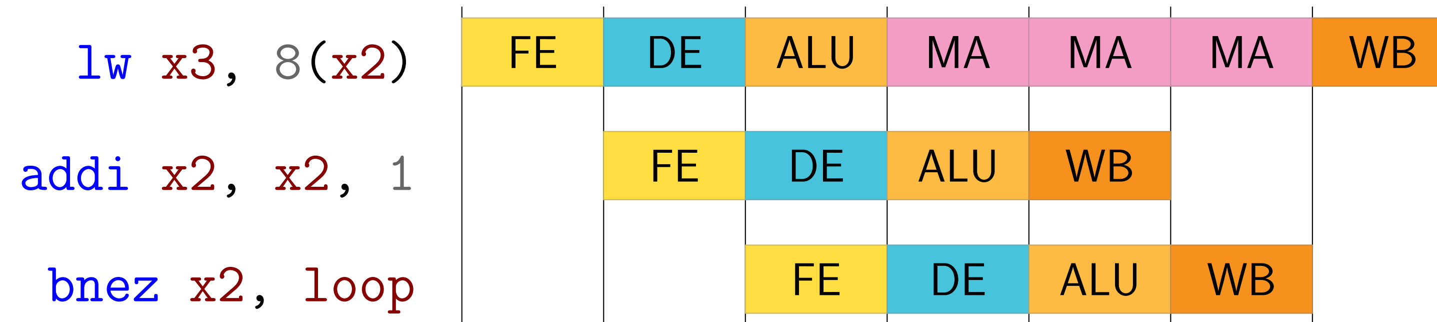
Problems?

Out-of-Order Completion

Even when started in correct order, instructions can complete *out-of-order*

Structural hazard on writeback stage, can be resolved

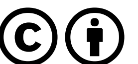
Example:



Problems?

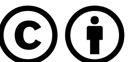
- What when there is an exception with the load?
- Example: Access fault, handled by OS, then continue

Out-of-Order Completion: Exception



Out-of-Order Completion: Exception

Problem:



Out-of-Order Completion: Exception

Problem:

- Following instructions completed when load exception occurs

Out-of-Order Completion: Exception

Problem:

- Following instructions completed when load exception occurs
- Exception is handled, for example by operating system

© ⓘ

Out-of-Order Completion: Exception

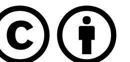
Problem:

- Following instructions completed when load exception occurs
- Exception is handled, for example by operating system
- Processing continues with re-issuing instructions starting with `lw`

Out-of-Order Completion: Exception

Problem:

- Following instructions completed when load exception occurs
- Exception is handled, for example by operating system
- Processing continues with re-issuing instructions starting with `lw`
- `addi` and `bnez` will be executed again, functional error



Out-of-Order Completion: Exception

Problem:

- Following instructions completed when load exception occurs
- Exception is handled, for example by operating system
- Processing continues with re-issuing instructions starting with `lw`
- `addi` and `bnez` will be executed again, functional error

Potential solutions

Out-of-Order Completion: Exception

Problem:

- Following instructions completed when load exception occurs
- Exception is handled, for example by operating system
- Processing continues with re-issuing instructions starting with `lw`
- `addi` and `bnez` will be executed again, functional error

Potential solutions

- Imprecise exceptions: The exception handler needs to clean up

CC BY

Out-of-Order Completion: Exception

Problem:

- Following instructions completed when load exception occurs
- Exception is handled, for example by operating system
- Processing continues with re-issuing instructions starting with `lw`
- `addi` and `bnez` will be executed again, functional error

Potential solutions

- Imprecise exceptions: The exception handler needs to clean up
- Start instruction processing only after sure no exception can occur

Out-of-Order Completion: Exception

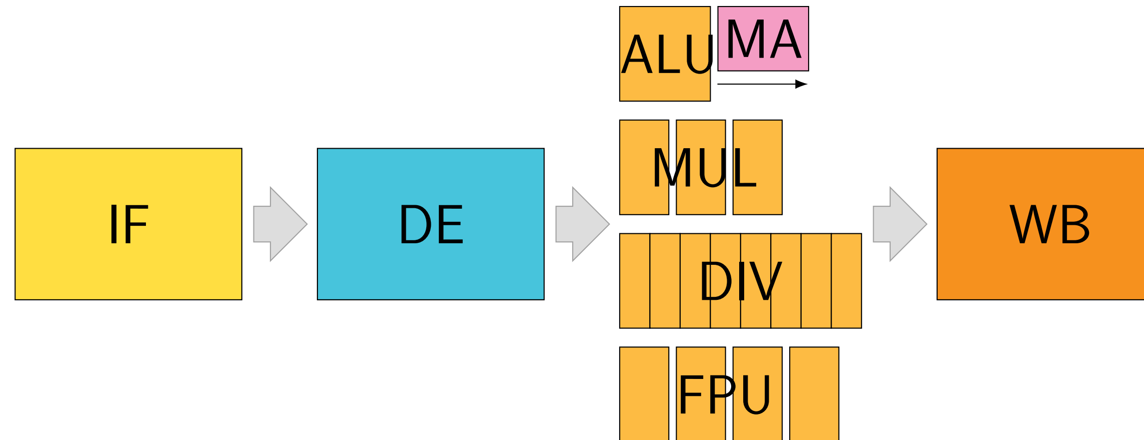
Problem:

- Following instructions completed when load exception occurs
- Exception is handled, for example by operating system
- Processing continues with re-issuing instructions starting with `lw`
- `addi` and `bnez` will be executed again, functional error

Potential solutions

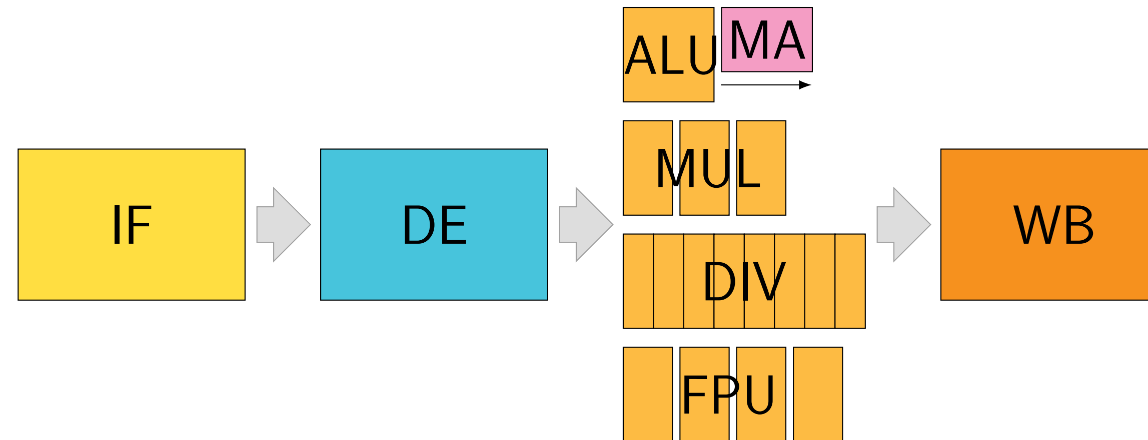
- Imprecise exceptions: The exception handler needs to clean up
- Start instruction processing only after sure no exception can occur
- Buffer results and commit in correct order (forwarding needs to look there too!)

Re-Order Buffer



Re-Order Buffer

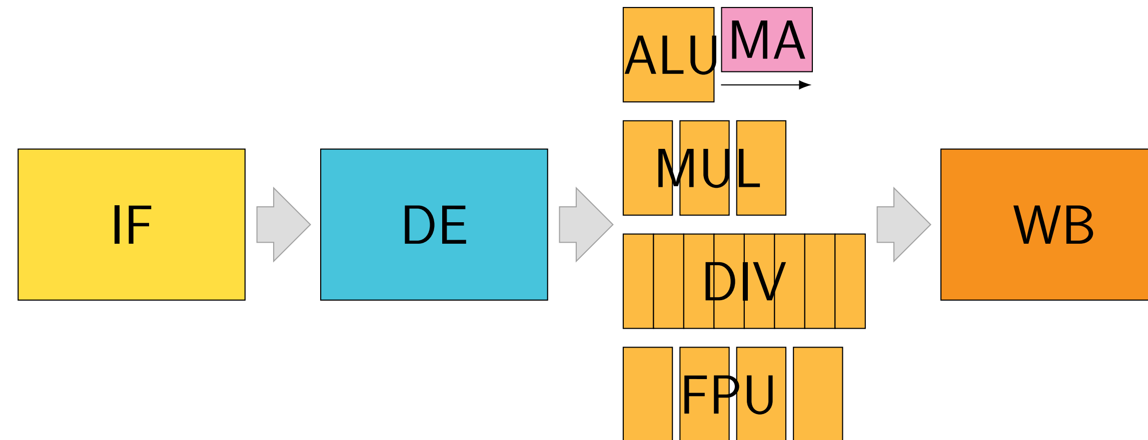
Split between instruction *retire* and architectural *commit*



Re-Order Buffer

Split between instruction *retire* and architectural *commit*

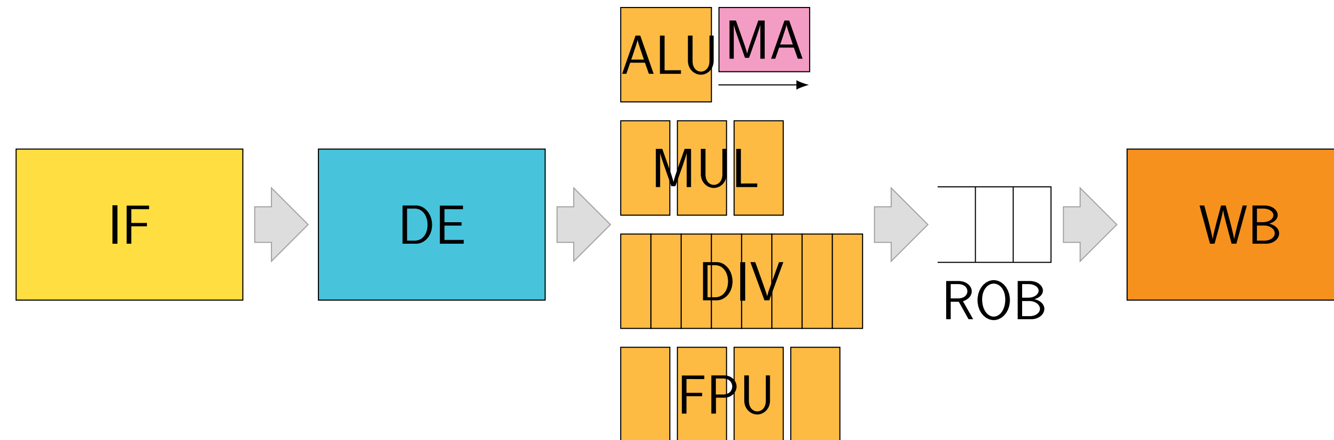
Re-order buffer (ROB) buffers results after out-of-order retire, commits in-order



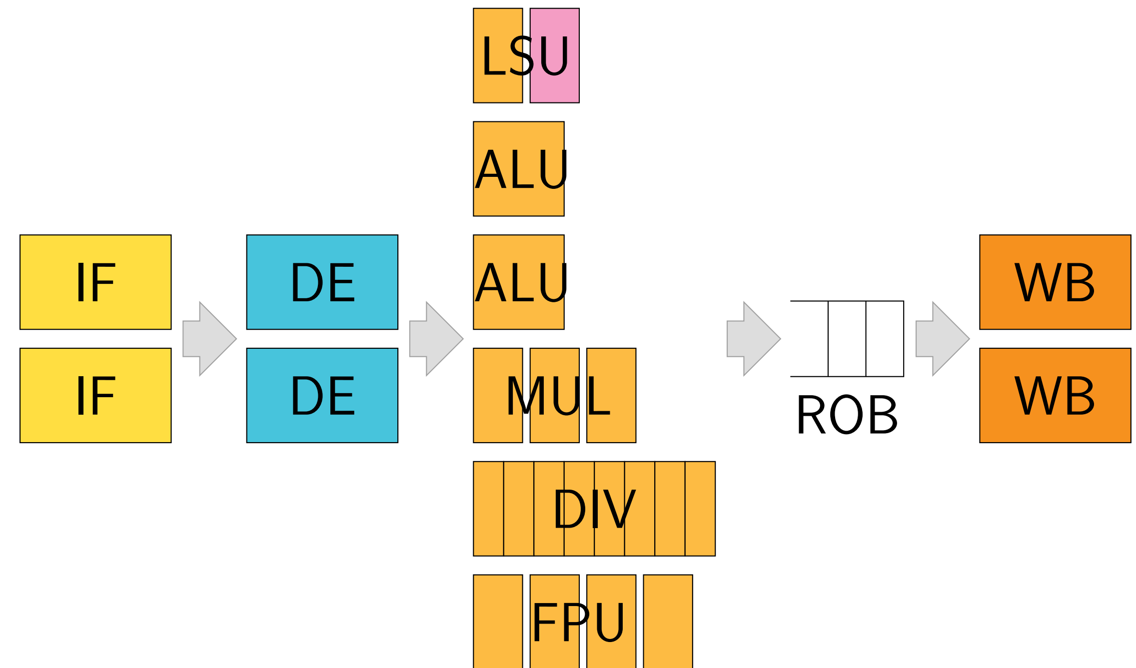
Re-Order Buffer

Split between instruction *retire* and architectural *commit*

Re-order buffer (ROB) buffers results after out-of-order retire, commits in-order

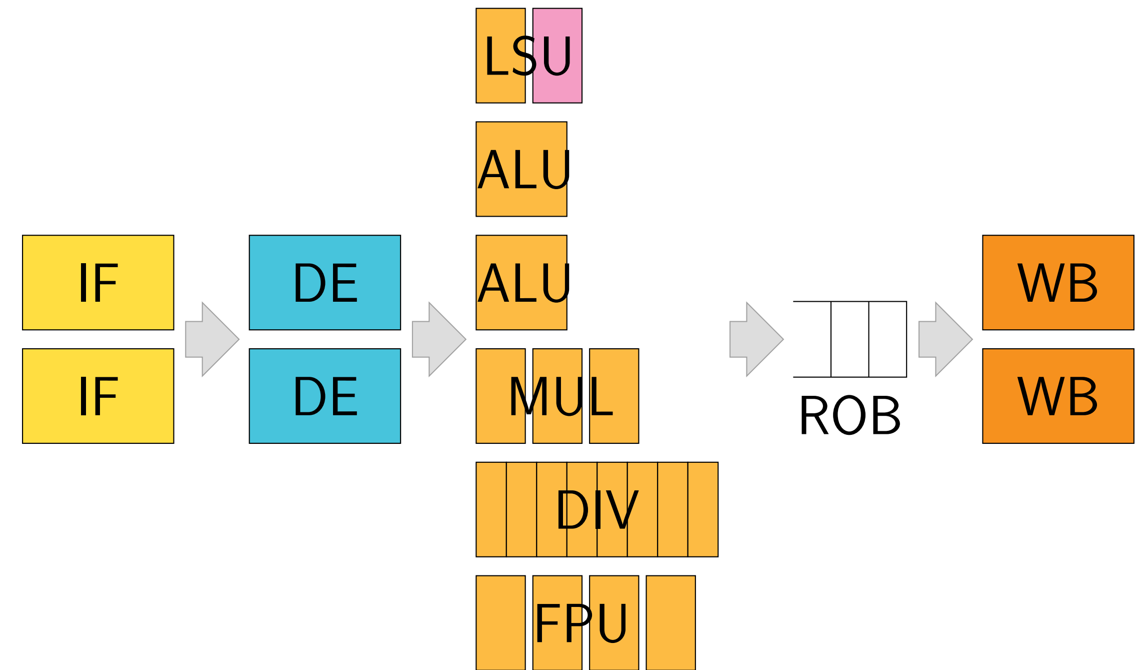


Superscalarity



Superscalarity

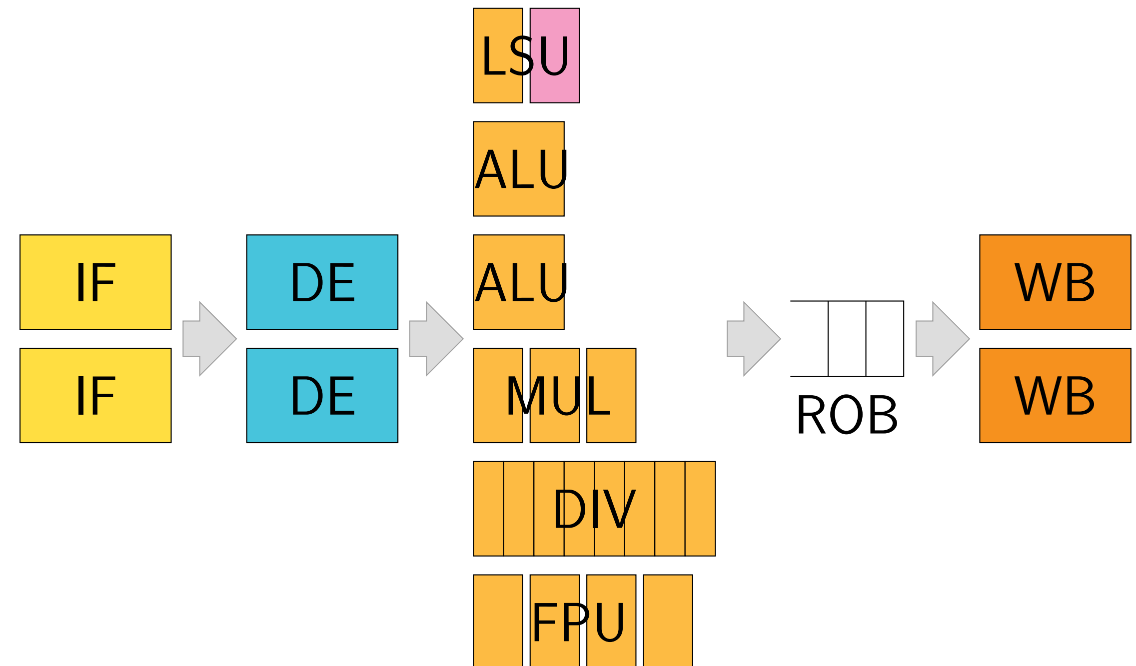
Execute multiple instructions in parallel



Superscalarity

Execute multiple instructions in parallel

Usually: replicate FUs

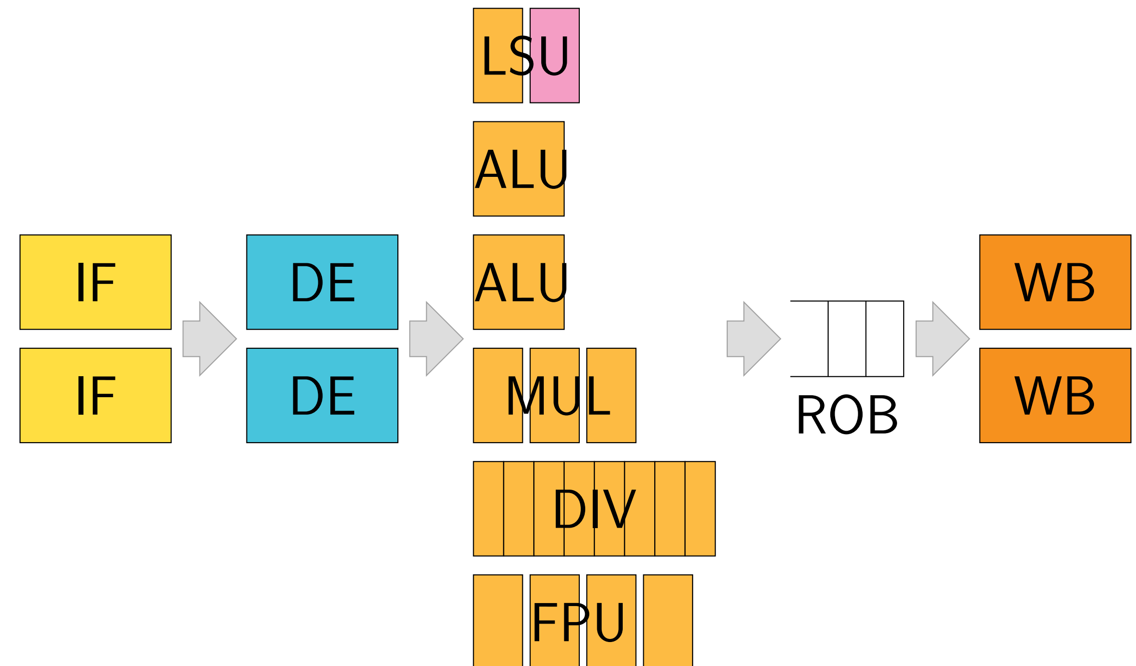


Superscalarity

Execute multiple instructions in parallel

Usually: replicate FUs

- ALU is often used

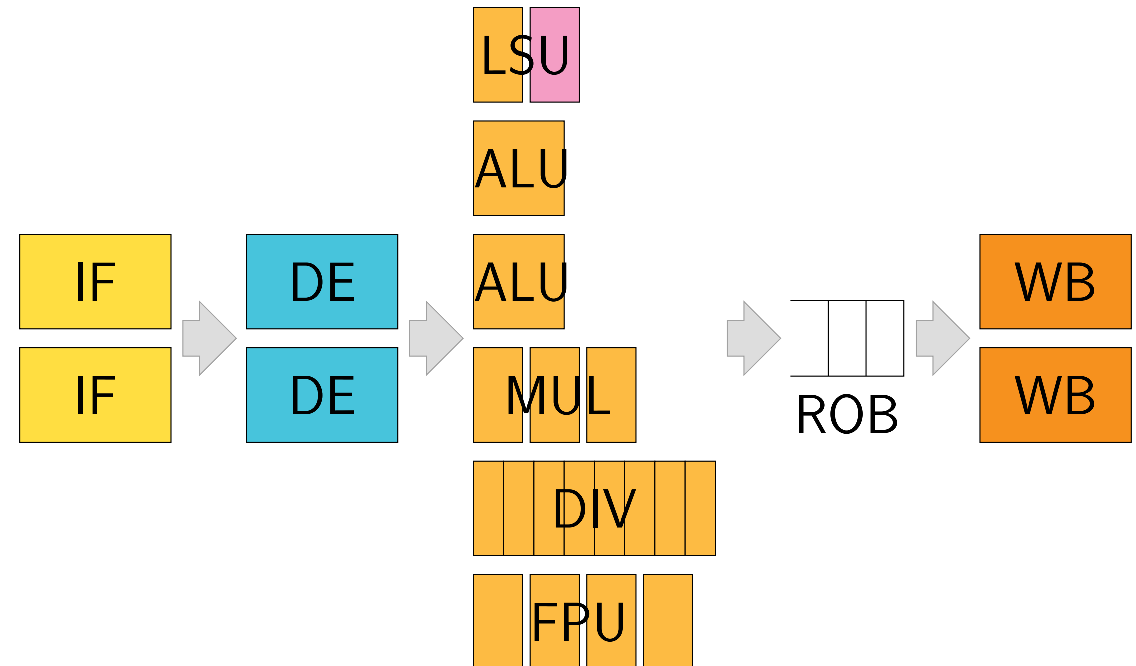


Superscalarity

Execute multiple instructions in parallel

Usually: replicate FUs

- ALU is often used
- LSU as separate FU



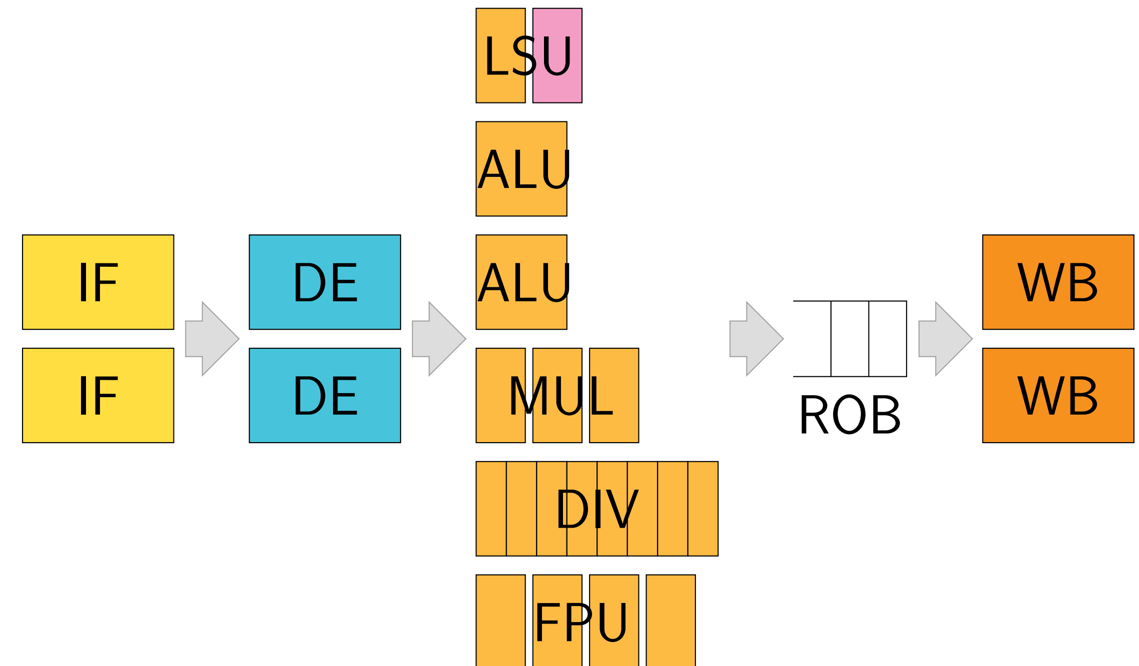
Superscalarity

Execute multiple instructions in parallel

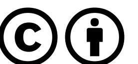
Usually: replicate FUs

- ALU is often used
- LSU as separate FU

Increases theoretical IPC by number of parallel instructions (**issue width**, here: 2)



Superscalarity: Example



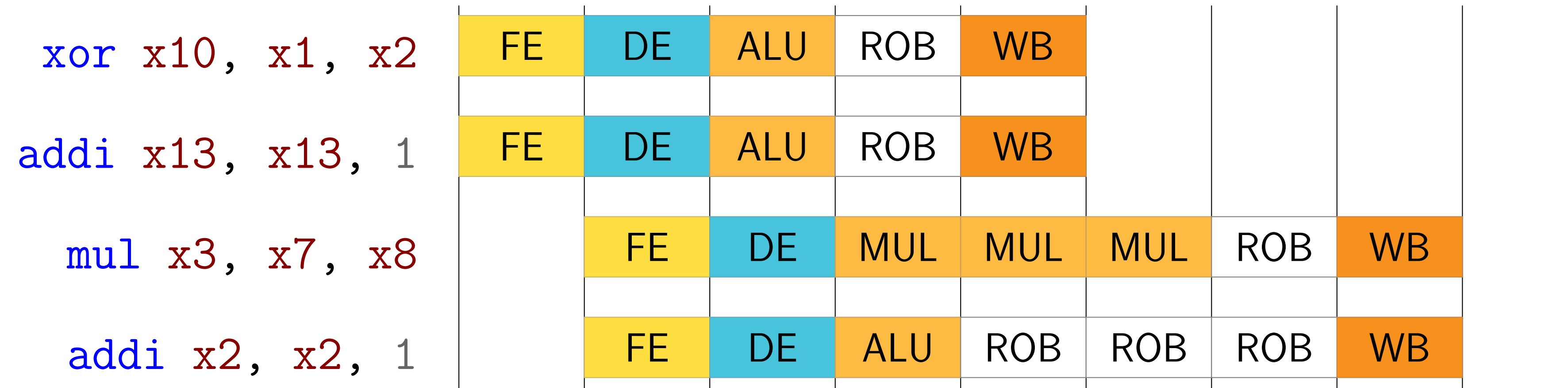
Superscalarity: Example

Exploit instruction level parallelism

Instruction	1	2	3	4	5	6	7	8	9
`xor x10, x1, x2`	FE	DE	ALU	ROB	WB				
`addi x13, x13, 1`	FE	DE	ALU	ROB	WB				
`mul x3, x7, x8`		FE	DE	MUL	MUL	MUL	ROB	WB	
`addi x2, x2, 1`		FE	DE	ALU	ROB	ROB	ROB	WB	

Superscalarity: Example

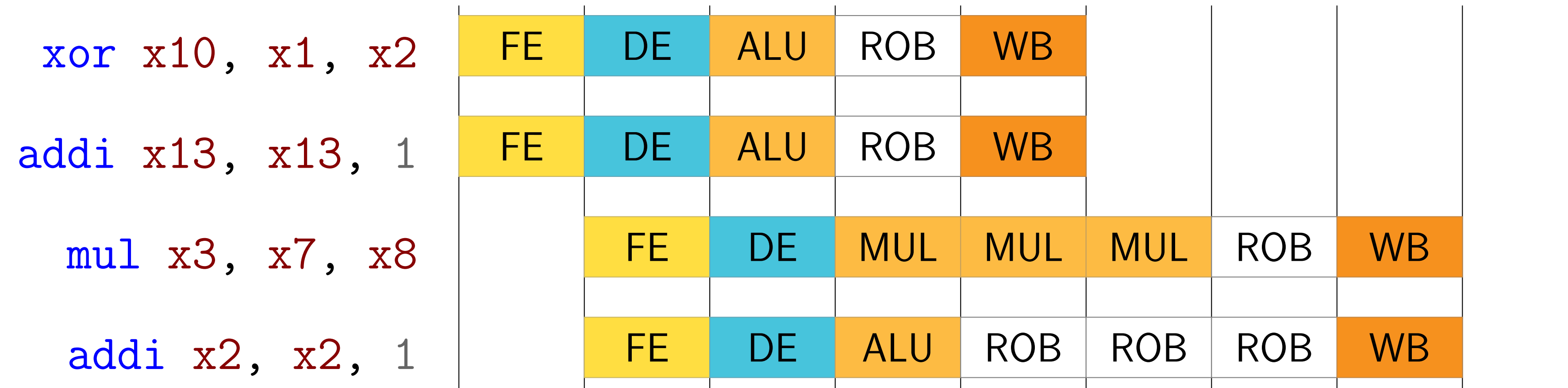
Exploit instruction level parallelism



Need issue width at each part of the pipeline, otherwise limits speedup

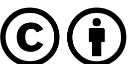
Superscalarity: Example

Exploit instruction level parallelism

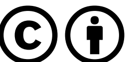


Need issue width at each part of the pipeline, otherwise limits speedup

Instruction stream split obvious here, but how do we schedule instructions in general?

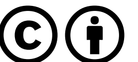


Instruction Scheduling



Instruction Scheduling

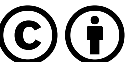
Scheduling: Select instructions to be started in EX stage



Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

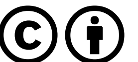
- From sequential order (as stored in memory)



Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies



Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies

Static Scheduling

Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies

Static Scheduling

- Execution of instructions pre-determined

Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies

Static Scheduling

- Execution of instructions pre-determined

Dynamic Scheduling

Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies

Static Scheduling

- Execution of instructions pre-determined

Dynamic Scheduling

- Selection of instructions at runtime

Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies

Static Scheduling

- Execution of instructions pre-determined

Dynamic Scheduling

- Selection of instructions at runtime
  - In-order (in sequential program order)

Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies

Static Scheduling

- Execution of instructions pre-determined

Dynamic Scheduling

- Selection of instructions at runtime
  - In-order (in sequential program order)
  - Out-of-order (whenever instructions are ready)

©ⓘ

Instruction Scheduling

Scheduling: Select instructions to be started in EX stage

- From sequential order (as stored in memory)
- Need to obey data dependencies

Static Scheduling

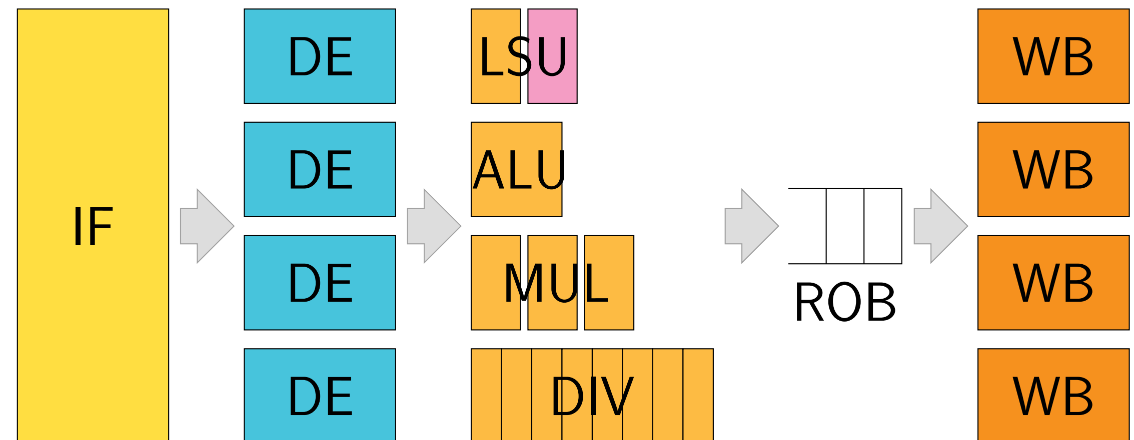
- Execution of instructions pre-determined

Dynamic Scheduling

- Selection of instructions at runtime
  - In-order (in sequential program order)
  - Out-of-order (whenever instructions are ready)

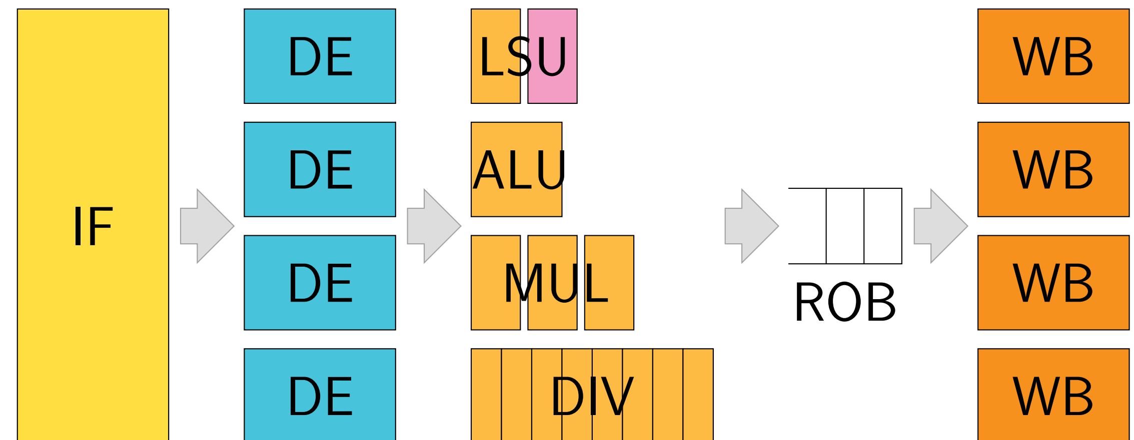
Scheduling and Superscalarity are independent concepts

Static Scheduling: VLIW



Static Scheduling: VLIW

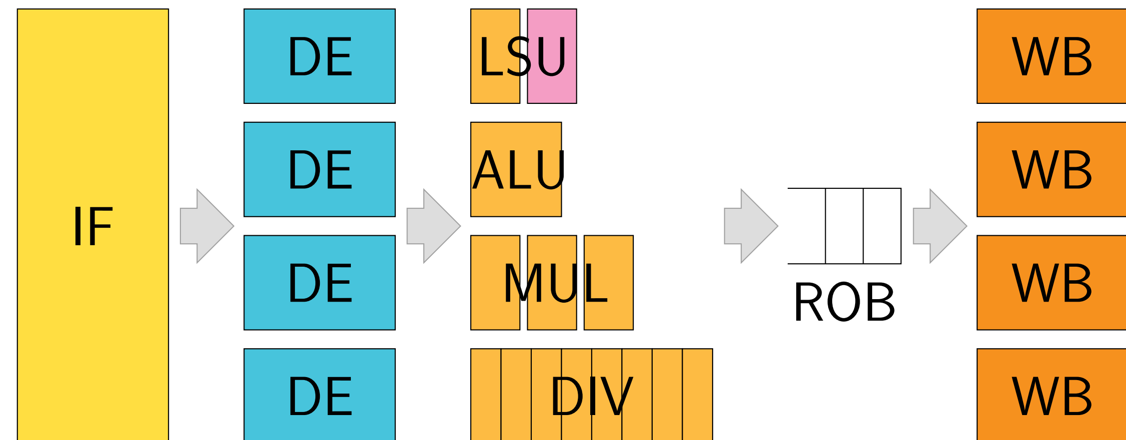
Very Long Instruction Word (VLIW)



Static Scheduling: VLIW

Very Long Instruction Word (VLIW)

Combine multiple instructions in slots of a long "packet"

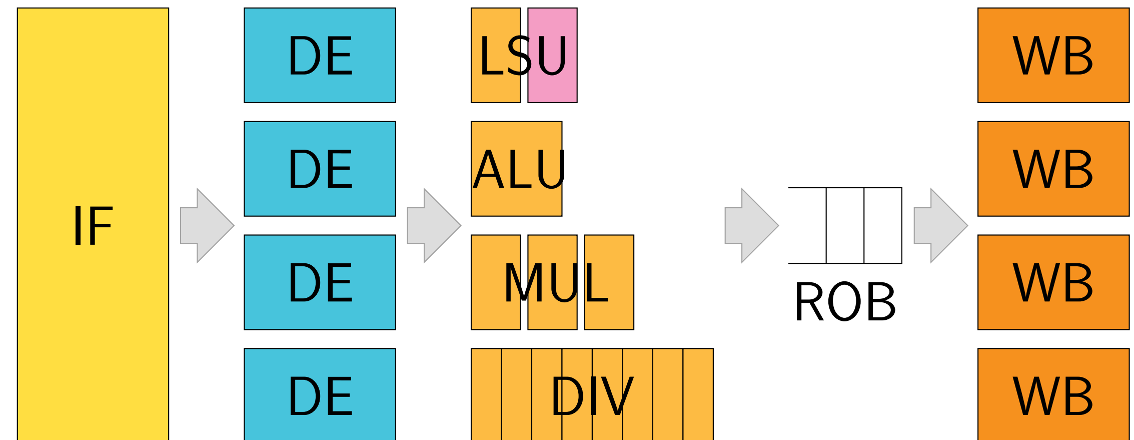


Static Scheduling: VLIW

Very Long Instruction Word (VLIW)

Combine multiple instructions in slots of a long "packet"

Start complete packet once all instructions in it can be started



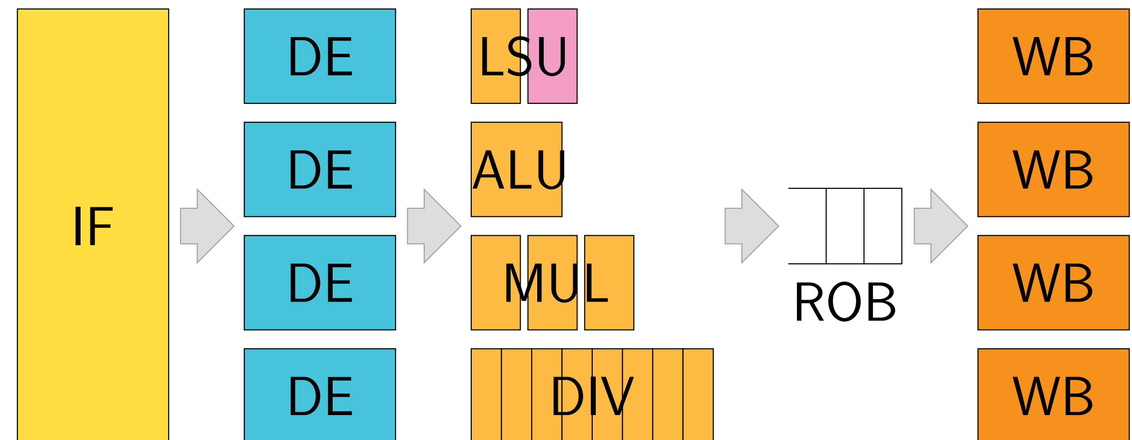
Static Scheduling: VLIW

Very Long Instruction Word (VLIW)

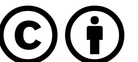
Combine multiple instructions in slots of a long "packet"

Start complete packet once all instructions in it can be started

Not much hardware overhead, but compilers are hard, challenge: use slots

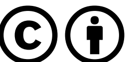


Dynamic Instruction Scheduling



Dynamic Instruction Scheduling

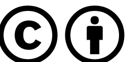
Approach: Start instructions as soon as possible



Dynamic Instruction Scheduling

Approach: Start instructions as soon as possible

- Once structural hazards are solved: functional unit is free



Dynamic Instruction Scheduling

Approach: Start instructions as soon as possible

- Once structural hazards are solved: functional unit is free
- Once data hazards are solved: dependencies resolved

Dynamic Instruction Scheduling

Approach: Start instructions as soon as possible

- Once structural hazards are solved: functional unit is free
- Once data hazards are solved: dependencies resolved

Keep next instruction(s) in buffer

Dynamic Instruction Scheduling

Approach: Start instructions as soon as possible

- Once structural hazards are solved: functional unit is free
- Once data hazards are solved: dependencies resolved

Keep next instruction(s) in buffer

- Instructions are still issued in order (FIFO instruction buffer)

Dynamic Instruction Scheduling

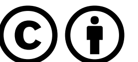
Approach: Start instructions as soon as possible

- Once structural hazards are solved: functional unit is free
- Once data hazards are solved: dependencies resolved

Keep next instruction(s) in buffer

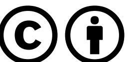
- Instructions are still issued in order (FIFO instruction buffer)
- Where to put it? (IF-DE or DE-EX)

Split the Decode Phase



Split the Decode Phase

Issue



Split the Decode Phase

Issue

- Decode instruction

Split the Decode Phase

Issue

- Decode instruction
- Check for structural hazards (can also be instruction buffer full)

Split the Decode Phase

Issue

- Decode instruction
- Check for structural hazards (can also be instruction buffer full)

Read Operands

Split the Decode Phase

Issue

- Decode instruction
- Check for structural hazards (can also be instruction buffer full)

Read Operands

- Executed after data dependencies are resolved

Split the Decode Phase

Issue

- Decode instruction
- Check for structural hazards (can also be instruction buffer full)

Read Operands

- Executed after data dependencies are resolved
- Then reads the operands

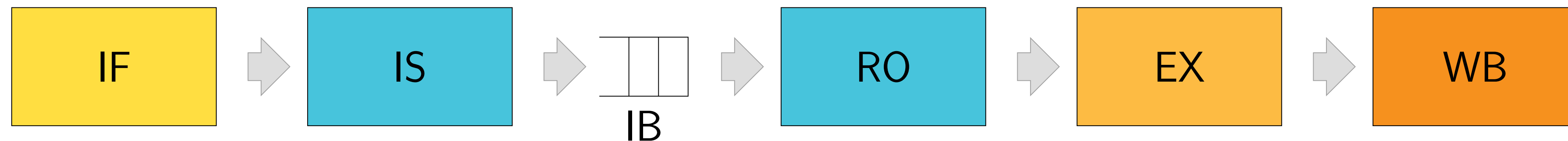
Split the Decode Phase

Issue

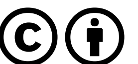
- Decode instruction
- Check for structural hazards (can also be instruction buffer full)

Read Operands

- Executed after data dependencies are resolved
- Then reads the operands

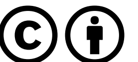


Scoreboard



Scoreboard

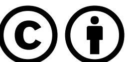
Scoreboard: Data structure to track execution



Scoreboard

Scoreboard: Data structure to track execution

Keeps all active instructions (in FUs)



Scoreboard

Scoreboard: Data structure to track execution

Keeps all active instructions (in FUs)

Check current instruction for conflict

Scoreboard

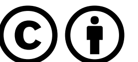
Scoreboard: Data structure to track execution

Keeps all active instructions (in FUs)

Check current instruction for conflict

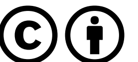
Instruction	FU	rd	rs1	rs2

Scoreboard



Scoreboard

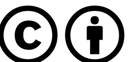
Check scoreboard for every instruction



Scoreboard

Check scoreboard for every instruction

- Structural hazard: Check if FU can start instruction



Scoreboard

Check scoreboard for every instruction

- Structural hazard: Check if FU can start instruction
- Read-after-Write: Check SB destination registers for instruction source registers

Scoreboard

Check scoreboard for every instruction

- Structural hazard: Check if FU can start instruction
- Read-after-Write: Check SB destination registers for instruction source registers
- Write-after-Read: Check SB source registers for instruction destination register

Scoreboard

Check scoreboard for every instruction

- Structural hazard: Check if FU can start instruction
- Read-after-Write: Check SB destination registers for instruction source registers
- Write-after-Read: Check SB source registers for instruction destination register
- Write-after-Write: Check SB destination register for instruction destination register

Scoreboard

Check scoreboard for every instruction

- Structural hazard: Check if FU can start instruction
- Read-after-Write: Check SB destination registers for instruction source registers
- Write-after-Read: Check SB source registers for instruction destination register
- Write-after-Write: Check SB destination register for instruction destination register

Lifecycle of entries

Scoreboard

Check scoreboard for every instruction

- Structural hazard: Check if FU can start instruction
- Read-after-Write: Check SB destination registers for instruction source registers
- Write-after-Read: Check SB source registers for instruction destination register
- Write-after-Write: Check SB destination register for instruction destination register

Lifecycle of entries

- Start instruction from instruction buffer once no hazards are left, add to SB

Scoreboard

Check scoreboard for every instruction

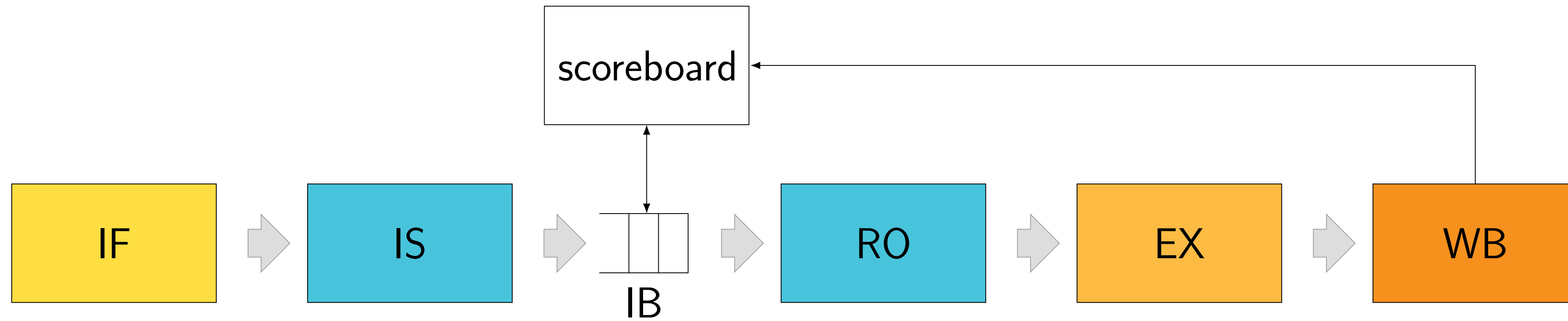
- Structural hazard: Check if FU can start instruction
- Read-after-Write: Check SB destination registers for instruction source registers
- Write-after-Read: Check SB source registers for instruction destination register
- Write-after-Write: Check SB destination register for instruction destination register

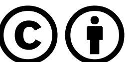
Lifecycle of entries

- Start instruction from instruction buffer once no hazards are left, add to SB
- Remove from SB once result was written

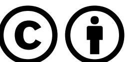
(C) (i)

Scoreboard: Integration



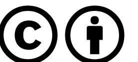


In-Order Scoreboard Scheduling



In-Order Scoreboard Scheduling

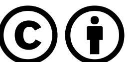
Use scoreboard for in-order dynamic scheduling



In-Order Scoreboard Scheduling

Use scoreboard for in-order dynamic scheduling

Typical: No bypassing (complexity of many functional units)

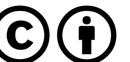


In-Order Scoreboard Scheduling

Use scoreboard for in-order dynamic scheduling

Typical: No bypassing (complexity of many functional units)

Scoreboard for in-order can be simplified



In-Order Scoreboard Scheduling

Use scoreboard for in-order dynamic scheduling

Typical: No bypassing (complexity of many functional units)

Scoreboard for in-order can be simplified

- Only the next instruction in buffer can be used

In-Order Scoreboard Scheduling

Use scoreboard for in-order dynamic scheduling

Typical: No bypassing (complexity of many functional units)

Scoreboard for in-order can be simplified

- Only the next instruction in buffer can be used

Can we have WAR and WAW for the next instruction?

In-Order Scoreboard Scheduling

Use scoreboard for in-order dynamic scheduling

Typical: No bypassing (complexity of many functional units)

Scoreboard for in-order can be simplified

- Only the next instruction in buffer can be used

Can we have WAR and WAW for the next instruction?

- **No WAR** as we are in order

In-Order Scoreboard Scheduling

Use scoreboard for in-order dynamic scheduling

Typical: No bypassing (complexity of many functional units)

Scoreboard for in-order can be simplified

- Only the next instruction in buffer can be used

Can we have WAR and WAW for the next instruction?

- **No WAR** as we are in order
- **WAW** can occur, hence need to track destination registers

In-Order Scoreboard Scheduling

Use scoreboard for in-order dynamic scheduling

Typical: No bypassing (complexity of many functional units)

Scoreboard for in-order can be simplified

- Only the next instruction in buffer can be used

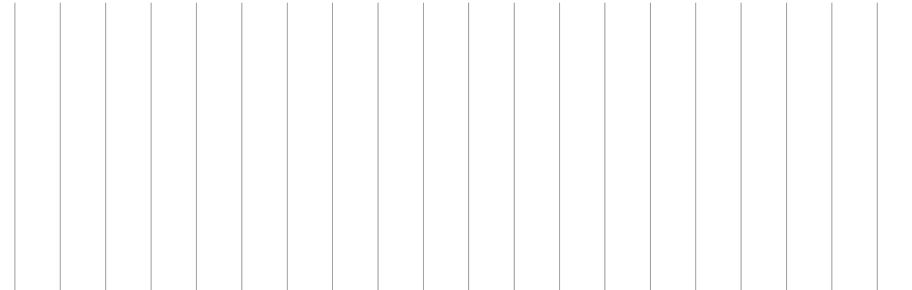
Can we have WAR and WAW for the next instruction?

- **No WAR** as we are in order
- **WAW** can occur, hence need to track destination registers
- We only need to know the pending writes (destination registers)

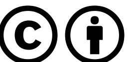
Scoreboard: Example

```
ld x12, 8(x9)
ld x13, 0(x7)
mul x17, x13, x12
subi x18, x12, 2
mul x13, x12, x18
add x10, x17, x13
```

Execution Diagram



Instruction	FU	rd

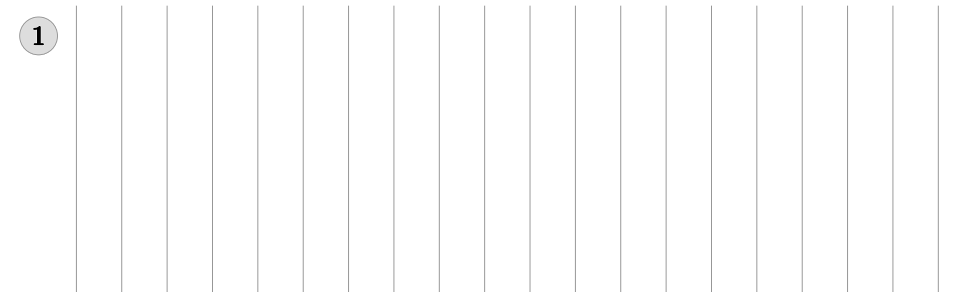


Scoreboard: Example

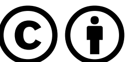
```
1 ld x12, 8(x9)
  ld x13, 0(x7)
  mul x17, x13, x12
  subi x18, x12, 2
  mul x13, x12, x18
  add x10, x17, x13
```

Instruction	FU	rd

Execution Diagram



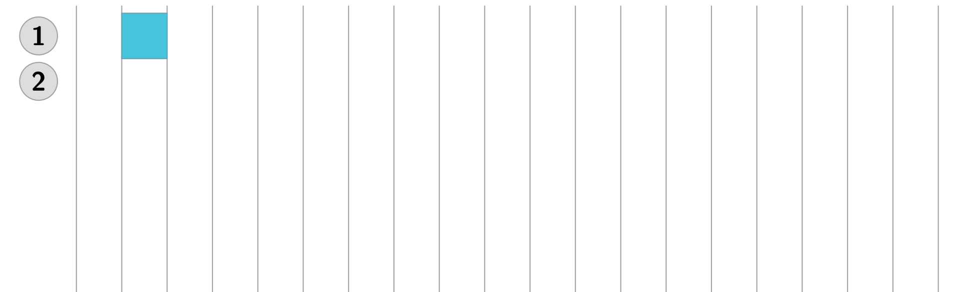
0: #1 can be started



Scoreboard: Example

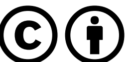
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
  mul x17, x13, x12
  subi x18, x12, 2
  mul x13, x12, x18
  add x10, x17, x13
```

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands

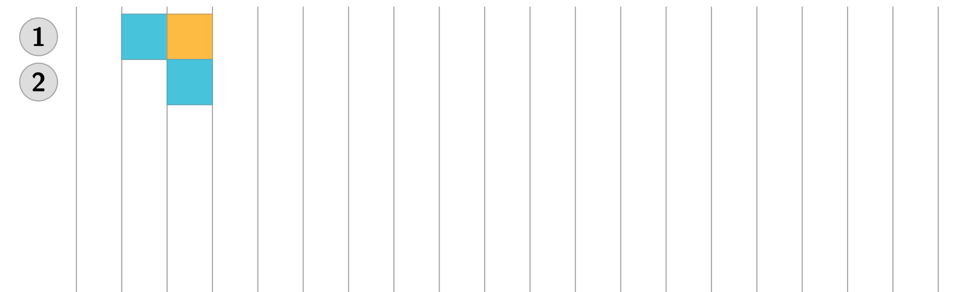
Instruction	FU	rd
1	LSU	x12



Scoreboard: Example

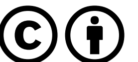
```
① ld x12, 8(x9)
② ld x13, 0(x7)
  mul x17, x13, x12
  subi x18, x12, 2
  mul x13, x12, x18
  add x10, x17, x13
```

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)

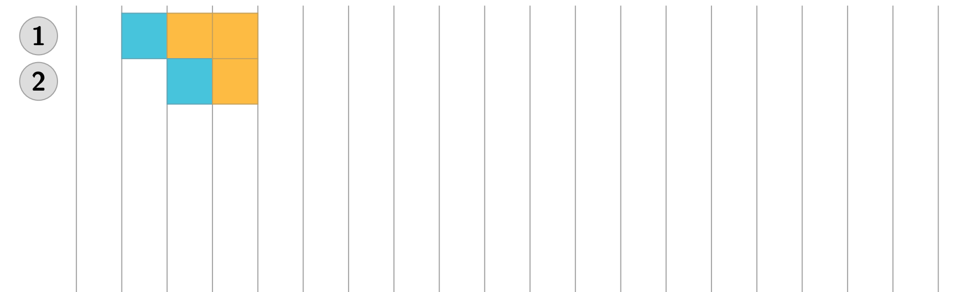
Instruction	FU	rd
1	LSU	x12
2	LSU	x13



Scoreboard: Example

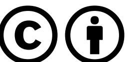
```
(1) ld x12, 8(x9)
(2) ld x13, 0(x7)
    mul x17, x13, x12
    subi x18, x12, 2
    mul x13, x12, x18
    add x10, x17, x13
```

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now

Instruction	FU	rd
1	LSU	x12
2	LSU	x13

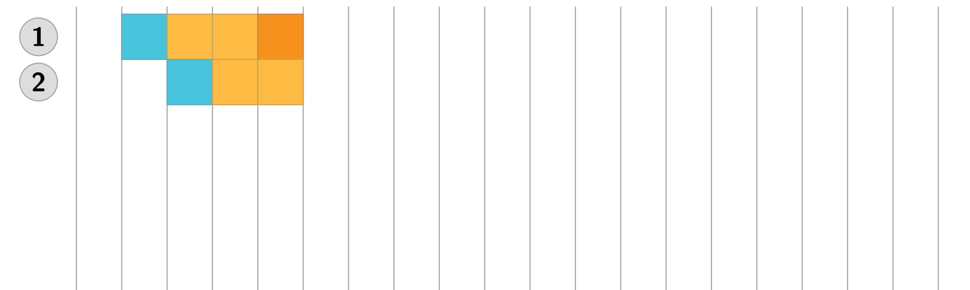


Scoreboard: Example

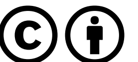
```
(1) ld x12, 8(x9)
(2) ld x13, 0(x7)
    mul x17, x13, x12
    subi x18, x12, 2
    mul x13, x12, x18
    add x10, x17, x13
```

Instruction	FU	rd
~~1~~	~~LSU~~	~~x12~~
2	LSU	x13

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too

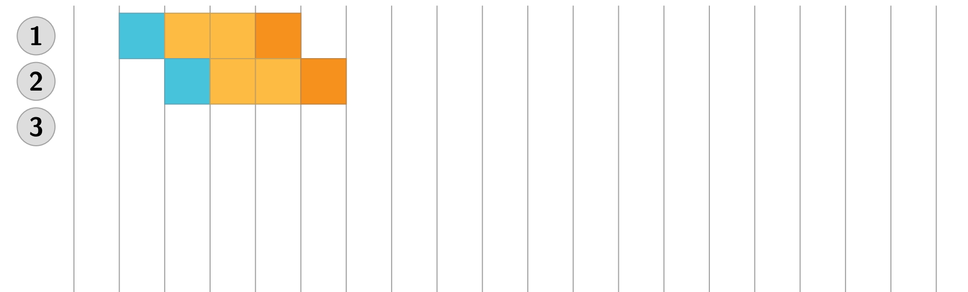


Scoreboard: Example

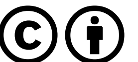
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
  subi x18, x12, 2
  mul x13, x12, x18
  add x10, x17, x13
```

Instruction	FU	rd
~~②~~	~~LSU~~	~~x13~~

Execution Diagram



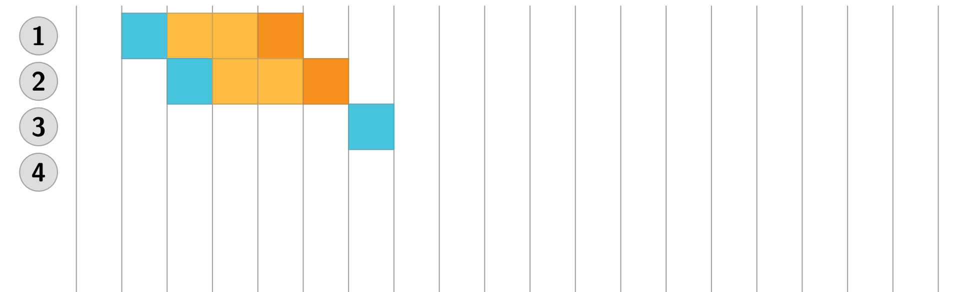
0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started



Scoreboard: Example

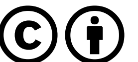
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
  mul x13, x12, x18
  add x10, x17, x13
```

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started

Instruction	FU	rd
③	MUL	x17



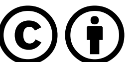
Scoreboard: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
  mul x13, x12, x18
  add x10, x17, x13
```

Instruction	FU	rd
3	MUL	x17
4	ALU	x18

Execution Diagram

0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked



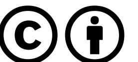
Scoreboard: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
  mul x13, x12, x18
  add x10, x17, x13
```

Execution Diagram

0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked

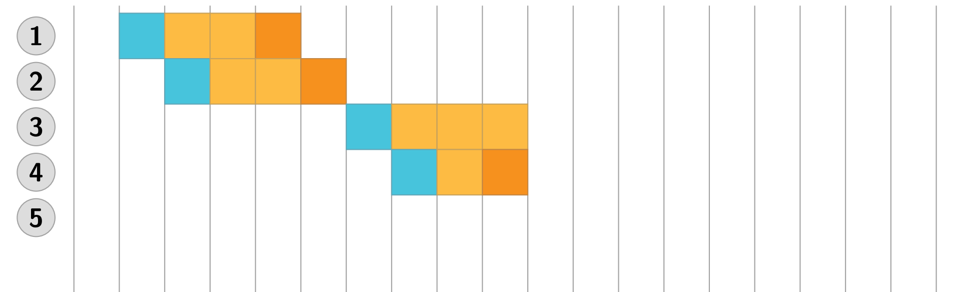
Instruction	FU	rd
3	MUL	x17
4	ALU	x18



Scoreboard: Example

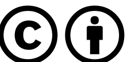
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
  add x10, x17, x13
```

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started

Instruction	FU	rd
3	MUL	x17
~~4~~	~~ALU~~	~~x18~~

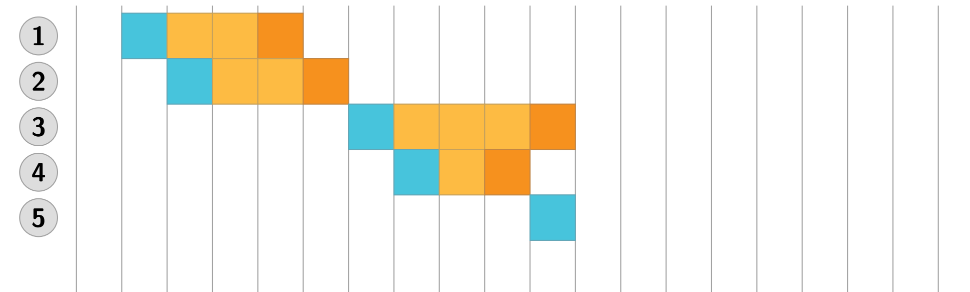


Scoreboard: Example

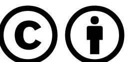
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
  add x10, x17, x13
```

Instruction	FU	rd
~~3~~	~~MUL~~	~~x17~~
5	MUL	x13

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked

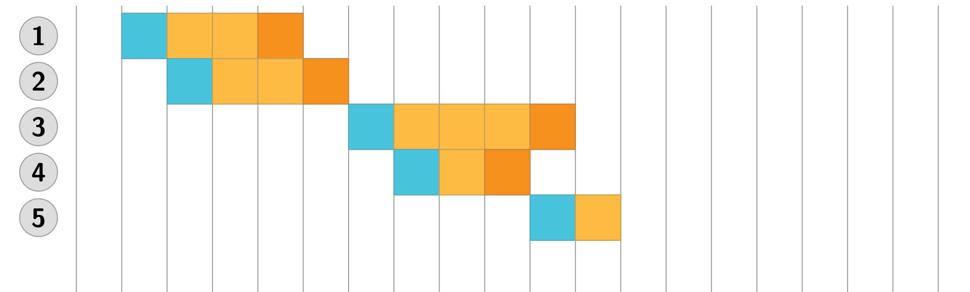


Scoreboard: Example

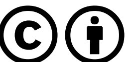
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
  add x10, x17, x13
```

Instruction	FU	rd
⑤	MUL	x13

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked

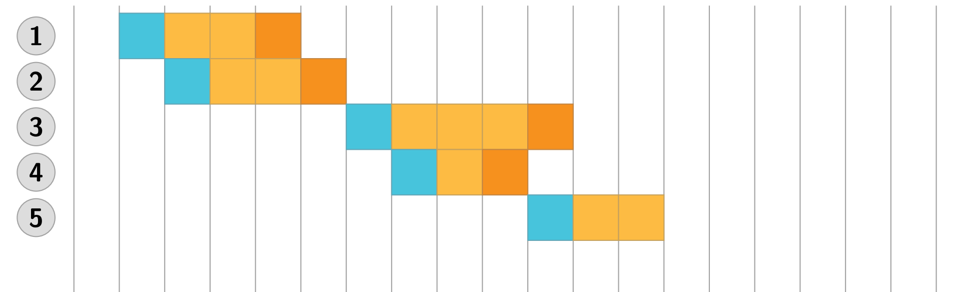


Scoreboard: Example

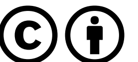
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
  add x10, x17, x13
```

Instruction	FU	rd
⑤	MUL	x13

Execution Diagram



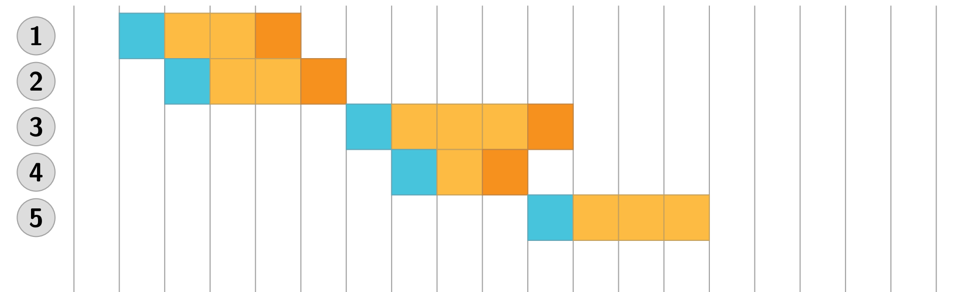
0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked



Scoreboard: Example

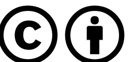
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
  add x10, x17, x13
```

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked

Instruction	FU	rd
5	MUL	x13

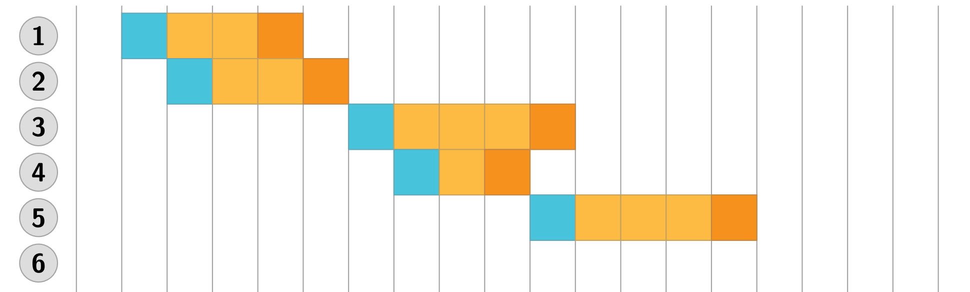


Scoreboard: Example

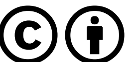
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

Instruction	FU	rd
~~5~~	~~MUL~~	~~x13~~

Execution Diagram



0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked
14: #5 in WB, #6 can be started

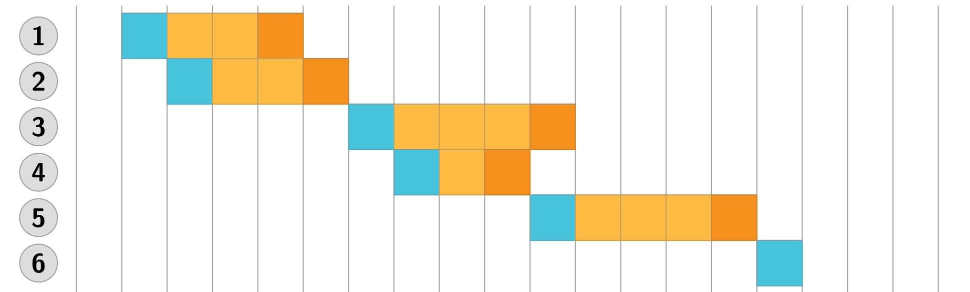


Scoreboard: Example

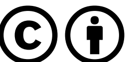
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

Instruction	FU	rd
6	ALU	x10

Execution Diagram



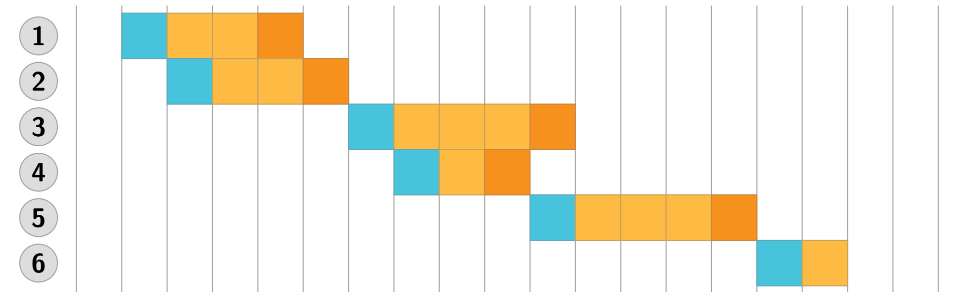
0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked
14: #5 in WB, #6 can be started



Scoreboard: Example

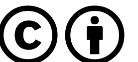
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

Execution Diagram



Instruction	FU	rd
6	ALU	x10

0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked
14: #5 in WB, #6 can be started

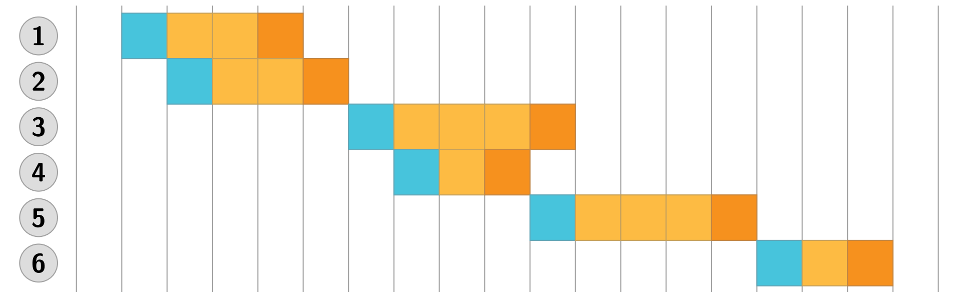


Scoreboard: Example

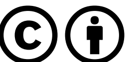
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

Instruction	FU	rd
~~6~~	~~ALU~~	~~x10~~

Execution Diagram

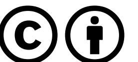


0: #1 can be started
1: #2 can be started, #1 reads operands
2: #3 is blocked (hazards)
3: #1 will complete now
4: #1 writes result, #2 will complete too
5: #2 in WB, #3 can be started
6: #4 can be started
7: #5 blocked
9: #4 in WB, #5 can be started
10: #6 is blocked
14: #5 in WB, #6 can be started



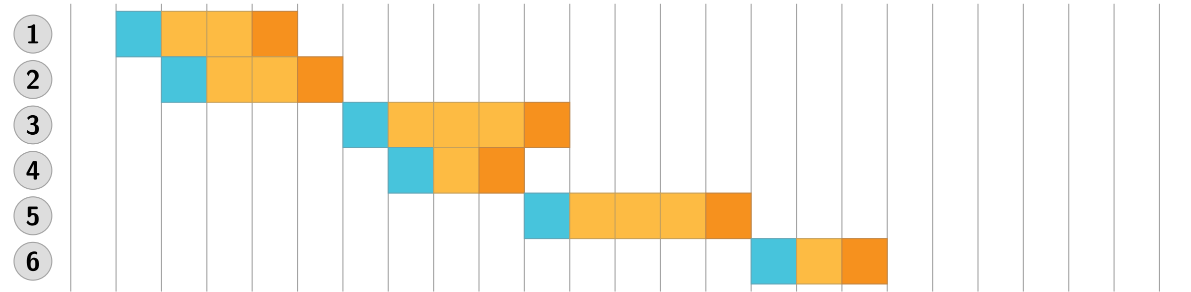
Example: Ariane CPU

https://github.com/pulp-platform/ariane/



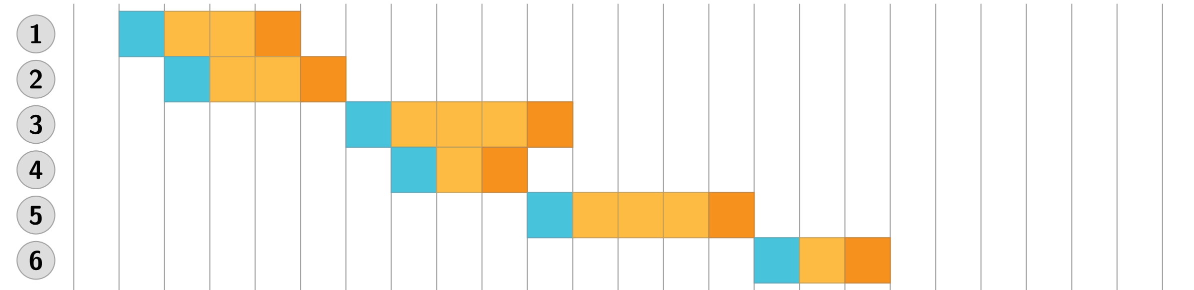
Limits of In-Order Issue

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

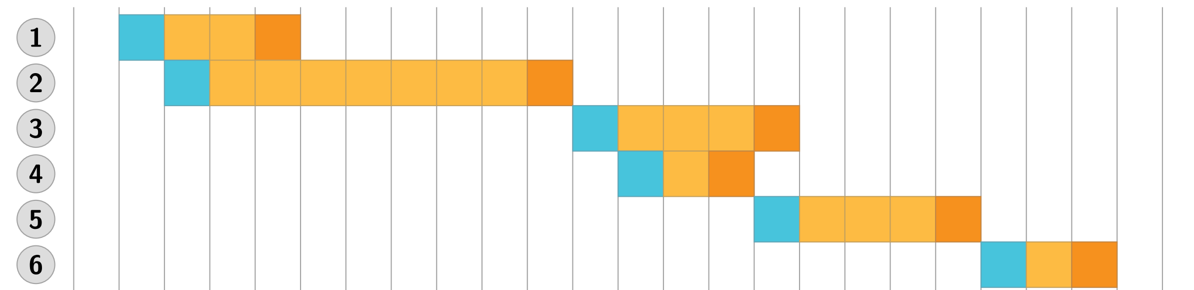


Limits of In-Order Issue

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```



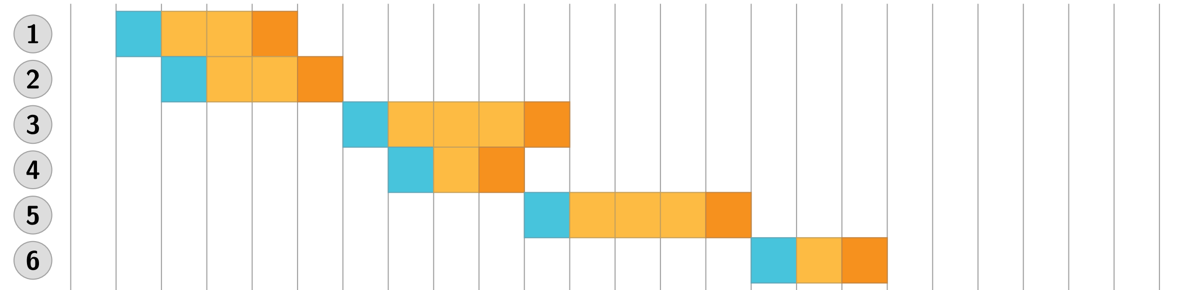
Delayed Load Instruction



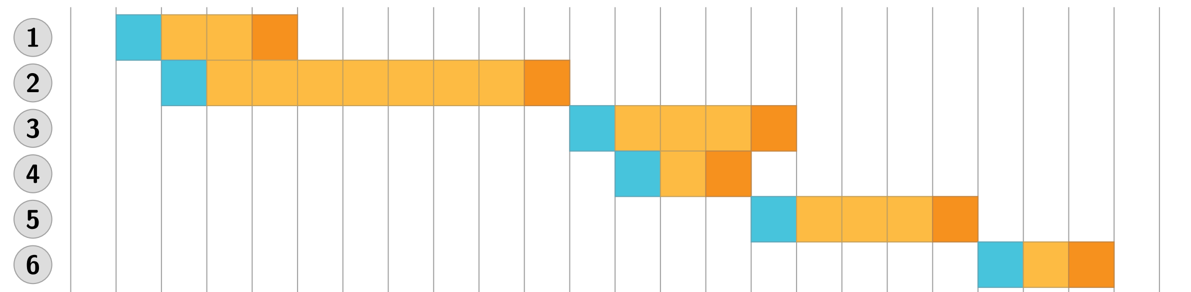
CC BY

Limits of In-Order Issue

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```



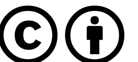
Delayed Load Instruction



Recap: data flow model, approach: reorder instructions

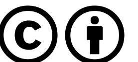
(c)(i)

Out-of-Order Issue



Out-of-Order Issue

Start of instructions in arbitrary order



Out-of-Order Issue

Start of instructions in arbitrary order

- As soon as hazards of each instruction are resolved

Out-of-Order Issue

Start of instructions in arbitrary order

- As soon as hazards of each instruction are resolved
- Instruction buffer has window of next N instructions

Out-of-Order Issue

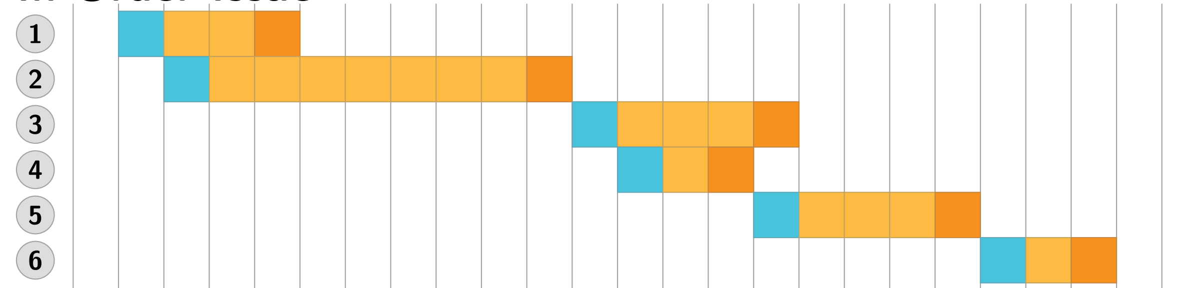
Start of instructions in arbitrary order

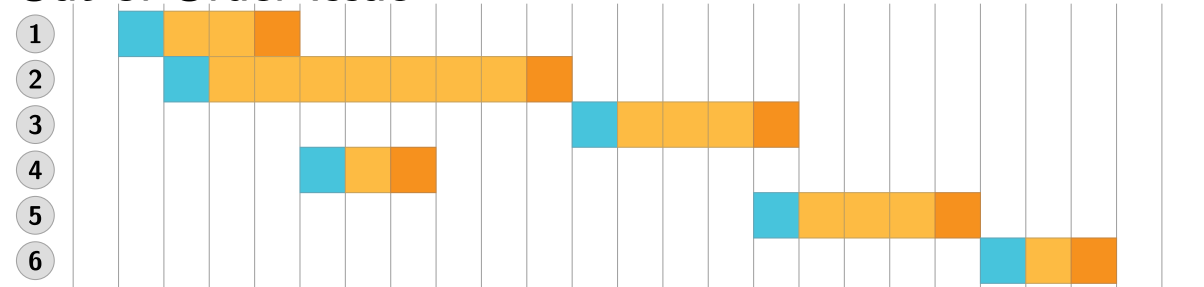
- As soon as hazards of each instruction are resolved
- Instruction buffer has window of next N instructions

But: Out-of-order issue does not improve very much alone!

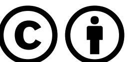
Out-of-Order: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```



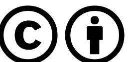


Out-of-Order Limits



Out-of-Order Limits

WAW and WAR limit further reordering



Out-of-Order Limits

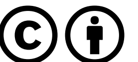
WAW and WAR limit further reordering

- Not real dependencies

Out-of-Order Limits

WAW and WAR limit further reordering

- Not real dependencies
- Artificially added: limitation of registers



Out-of-Order Limits

WAW and WAR limit further reordering

- Not real dependencies
- Artificially added: limitation of registers

Problem with limited registers

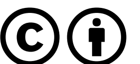
Out-of-Order Limits

WAW and WAR limit further reordering

- Not real dependencies
- Artificially added: limitation of registers

Problem with limited registers

- Number of registers limited by ISA



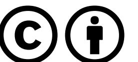
Out-of-Order Limits

WAW and WAR limit further reordering

- Not real dependencies
- Artificially added: limitation of registers

Problem with limited registers

- Number of registers limited by ISA
- Compiler optimizations limited



Out-of-Order Limits

WAW and WAR limit further reordering

- Not real dependencies
- Artificially added: limitation of registers

Problem with limited registers

- Number of registers limited by ISA
- Compiler optimizations limited
- Especially with different execution paths

Out-of-Order Limits

WAW and WAR limit further reordering

- Not real dependencies
- Artificially added: limitation of registers

Problem with limited registers

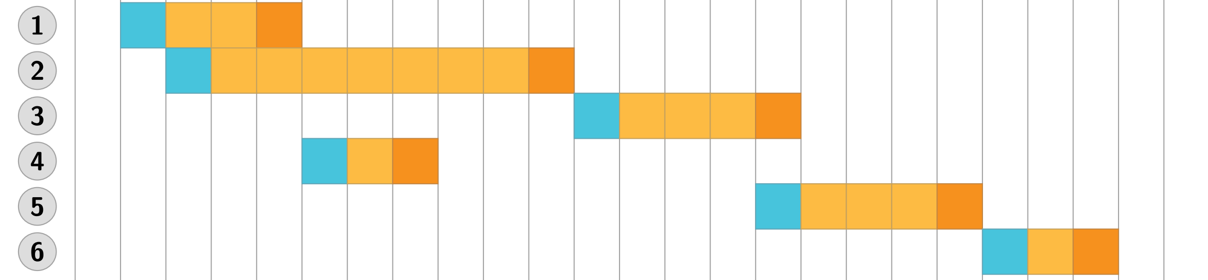
- Number of registers limited by ISA
- Compiler optimizations limited
- Especially with different execution paths

Approach: CPU solves problem by register renaming

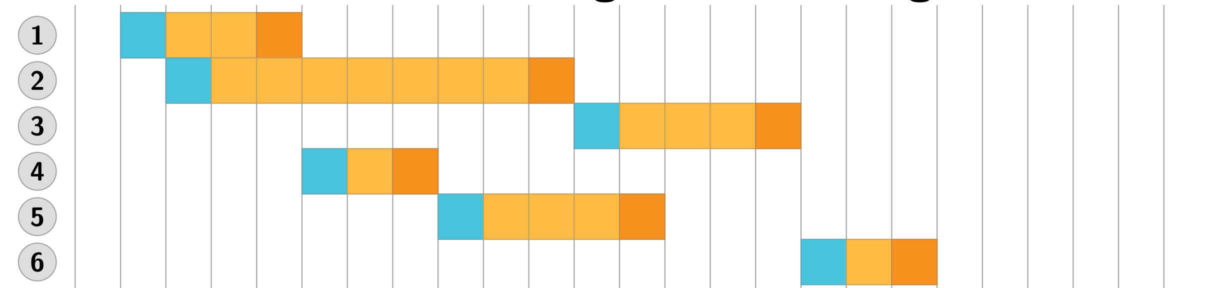
Register Renaming: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

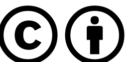
Out-of-Order Issue



Out-of-Order Issue with Register Renaming

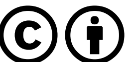


Register Renaming



Register Renaming

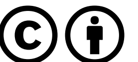
Approach: Use microarchitecture ("virtual" register names)



Register Renaming

Approach: Use microarchitecture (”virtual” register names)

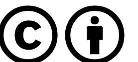
- Entirely eliminates WAR and WAW hazards



Register Renaming

Approach: Use microarchitecture ("virtual" register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world



Register Renaming

Approach: Use microarchitecture ("virtual" register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

Register Renaming

Approach: Use microarchitecture ("virtual" register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

- Reservation stations store instructions and renames

Register Renaming

Approach: Use microarchitecture ("virtual" register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

- Reservation stations store instructions and renames
- Format of reservation stations (multiple entries per FU)

Register Renaming

Approach: Use microarchitecture ("virtual" register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

- Reservation stations store instructions and renames
- Format of reservation stations (multiple entries per FU)
  - Op: Operation

Register Renaming

Approach: Use microarchitecture (”virtual” register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

- Reservation stations store instructions and renames
- Format of reservation stations (multiple entries per FU)
  - Op: Operation
  - Qj, Qk: Reservation station that produces source registers (pending)

CC BY

Register Renaming

Approach: Use microarchitecture (”virtual” register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

- Reservation stations store instructions and renames
- Format of reservation stations (multiple entries per FU)
  - Op: Operation
  - Qj, Qk: Reservation station that produces source registers (pending)
  - Vj, Vk: Value of source register (once available)

Register Renaming

Approach: Use microarchitecture (”virtual” register names)

- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

- Reservation stations store instructions and renames
- Format of reservation stations (multiple entries per FU)
  - Op: Operation
  - Qj, Qk: Reservation station that produces source registers (pending)
  - Vj, Vk: Value of source register (once available)
  - Busy: Reservation station is active

Register Renaming

Approach: Use microarchitecture ("virtual" register names)

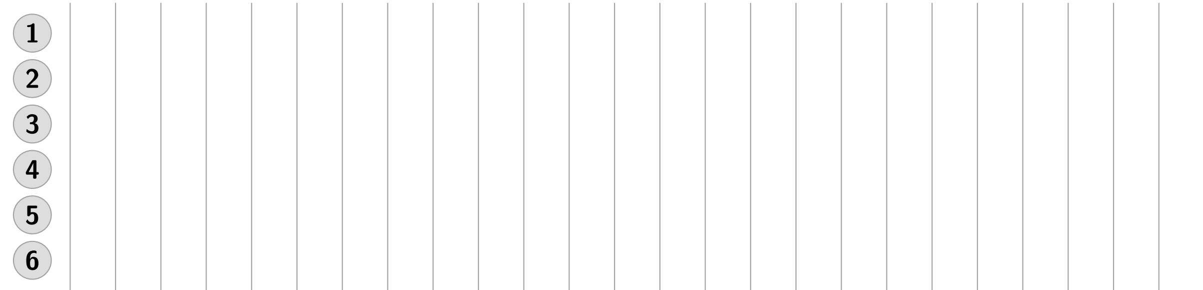
- Entirely eliminates WAR and WAW hazards
- Not visible to the outside world

Introduced by Robert Tomasulo (1967)

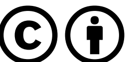
- Reservation stations store instructions and renames
- Format of reservation stations (multiple entries per FU)
  - Op: Operation
  - Qj, Qk: Reservation station that produces source registers (pending)
  - Vj, Vk: Value of source register (once available)
  - Busy: Reservation station is active
- Additionally: Register result status shows which RS produces registers

Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

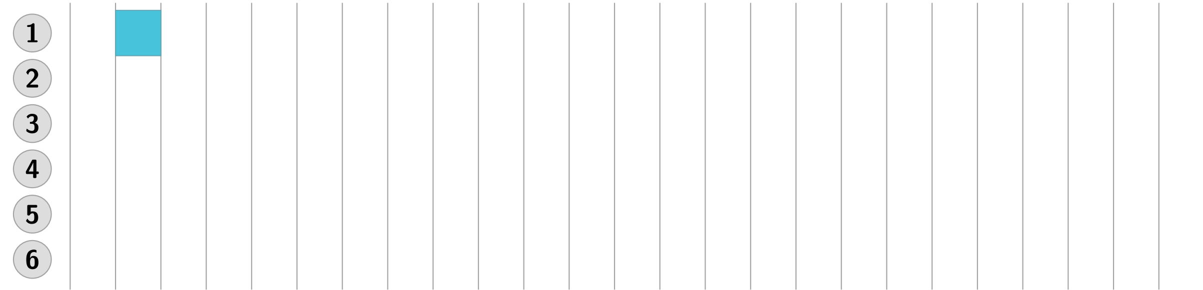


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0															
1															

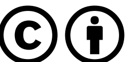


Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

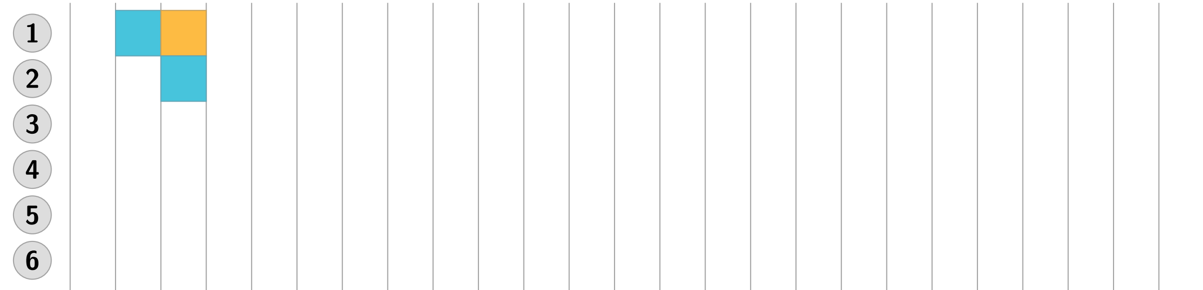


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0						1	8	...	-	-					
1															



Tomasulo: Example

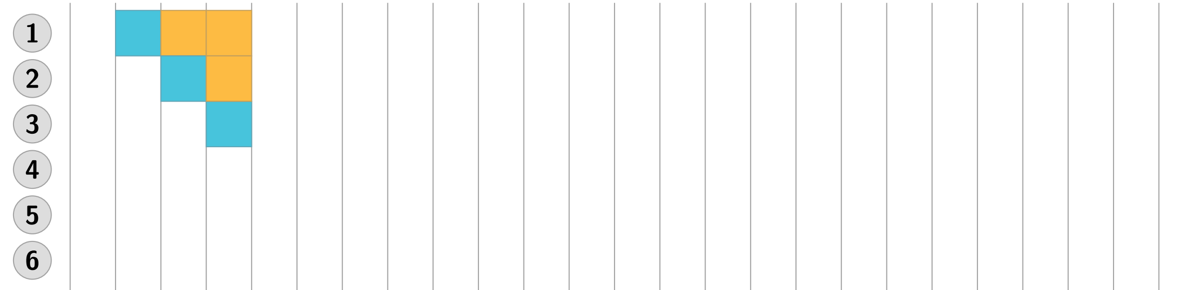
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```



	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0						1	8	...	-	-					
1						2	0	...	-	-					

Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

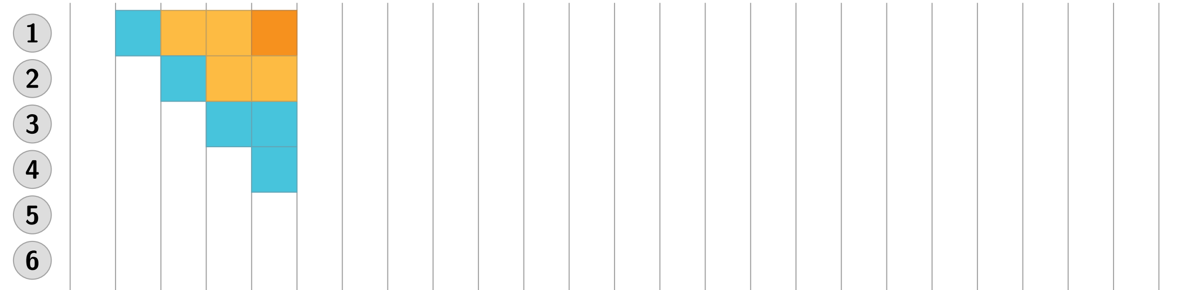


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0						1	8	...	-	-	3	-	-	LSU0	LSU1
1						2	0	...	-	-					

Prof. Dr.-Ing. Stefan Wallentowitz – Department 07 – Munich University of Applied Sciences

Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

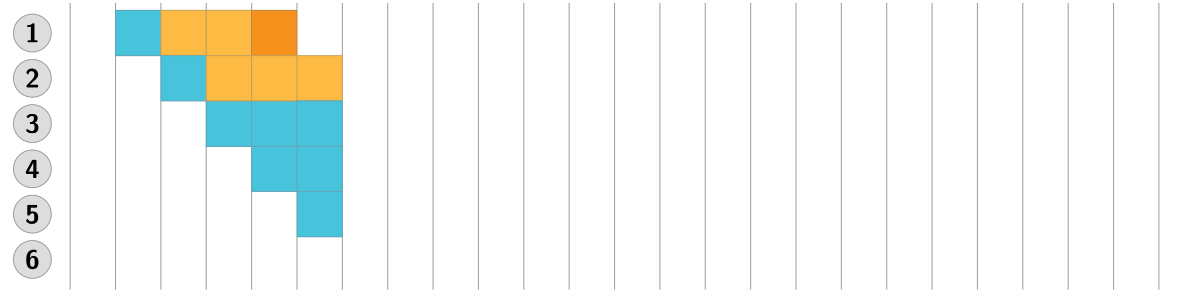


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0	4	-	2	LSU0	-	1	8	...	-	-	3	-	-	LSU0	LSU1
1						2	0	...	-	-					

CC BY

Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

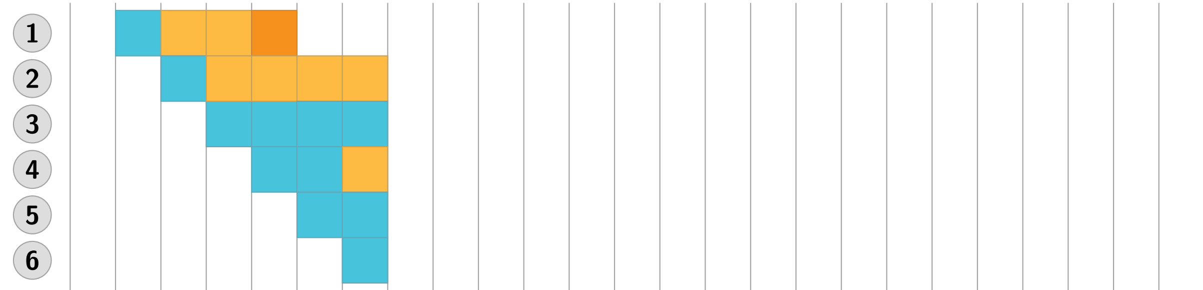


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0	4	...	2	~~LSU0~~	-	~~1~~	~~8~~	~~...~~	-	-	3	...	-	~~LSU0~~	LSU1
1						2	0	...	-	-	5	...	-	-	ALU0

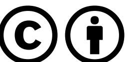
(cc) (by)

Tomasulo: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

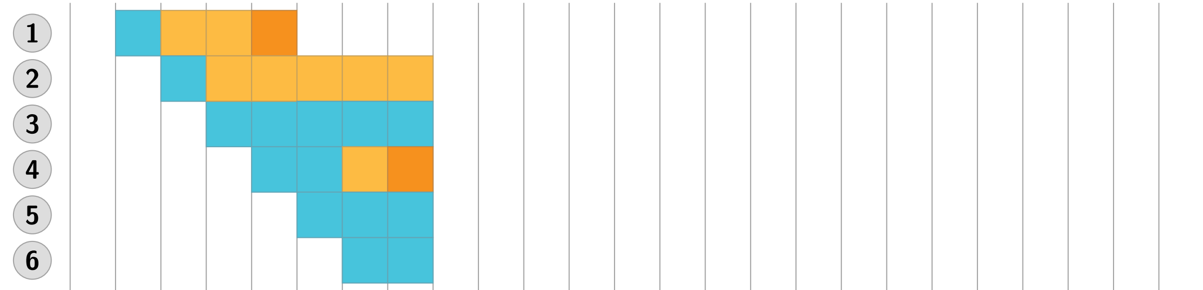


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0	4	...	2	-	-						3	...	-	-	LSU1
1	6	-	-	MUL0	MUL1	2	0	...	-	-	5	...	-	-	ALU0



Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```



	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0	4	...	2	-	-						3	...	-	-	LSU1
1	6	-	-	MUL0	MUL1	2	0	...	-	-	5	...	-	-	ALU0

(c)(i)

Tomasulo: Example

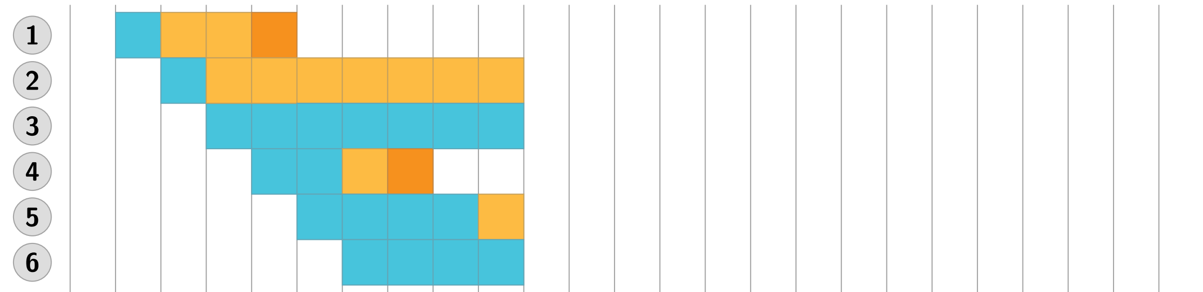
```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0	4	~~...~~	~~2~~	-	-						3	...	-	-	LSU1
1	6	-	-	MUL0	MUL1	2	0	...	-	-	5	...	...	-	~~ALU0~~

CC BY

Tomasulo: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

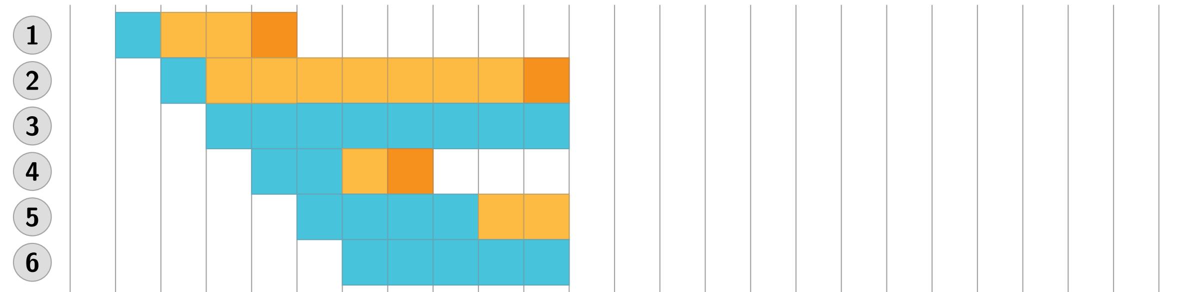


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											3	...	-	-	LSU1
1	6	-	-	MUL0	MUL1	2	0	...	-	-	5	...	...	-	-

CC BY

Tomasulo: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

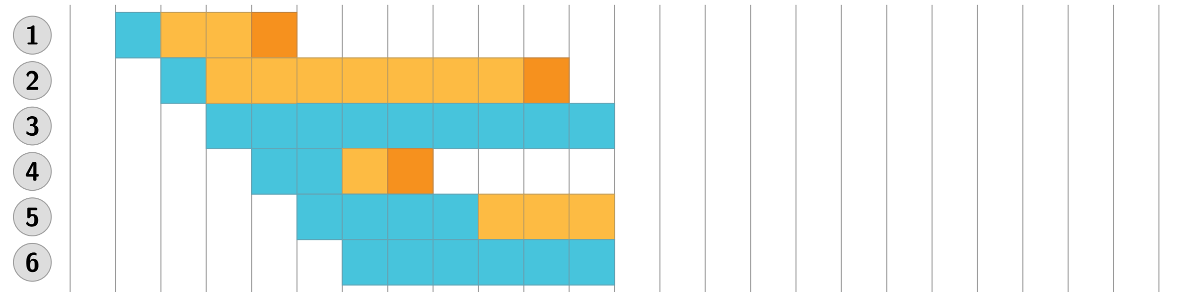


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											3	...	-	-	LSU1
1	6	-	-	MUL0	MUL1	2	0	...	-	-	5	...	...	-	-

CC BY

Tomasulo: Example

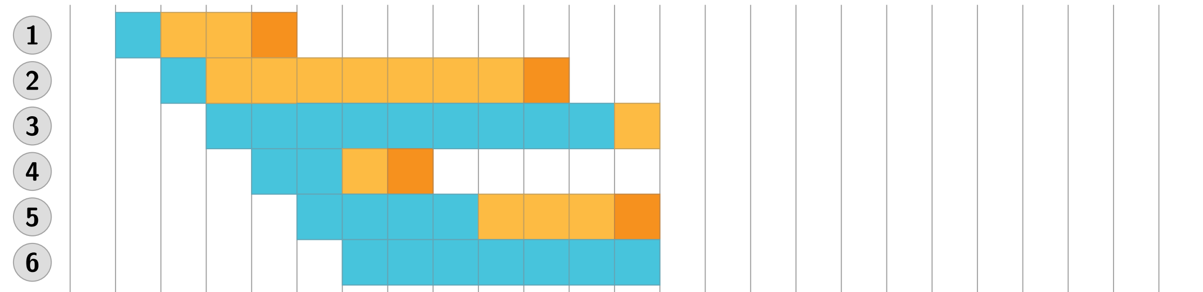
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```



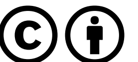
	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											3	…	…	-	~~LSU1~~
1	6	-	-	MUL0	MUL1	~~2~~	~~0~~	~~…~~	-	-	5	…	…	-	-

Tomasulo: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

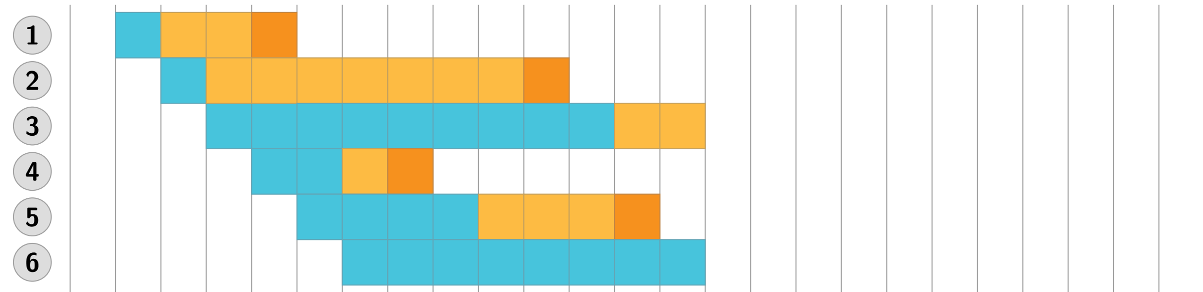


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											3	...	...	-	-
1	6	-	-	MUL0	MUL1						5	...	...	-	-



Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

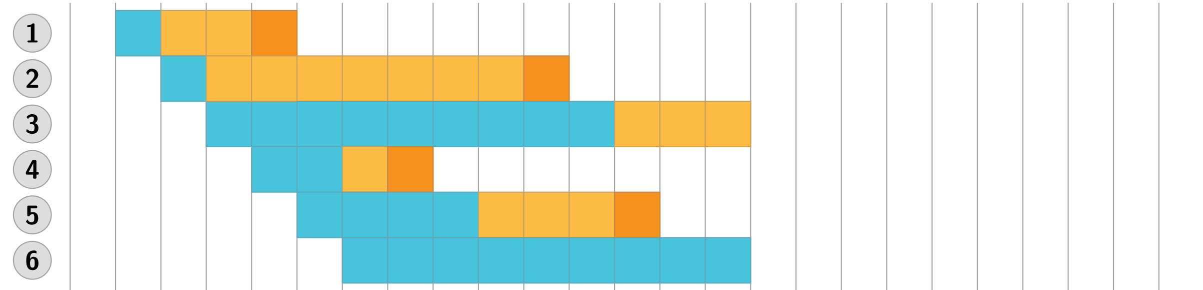


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											3	...	...	-	-
1	6	-	...	MUL0	~~MUL1~~						~~5~~	~~...~~	~~...~~	-	-

(cc) (by)

Tomasulo: Example

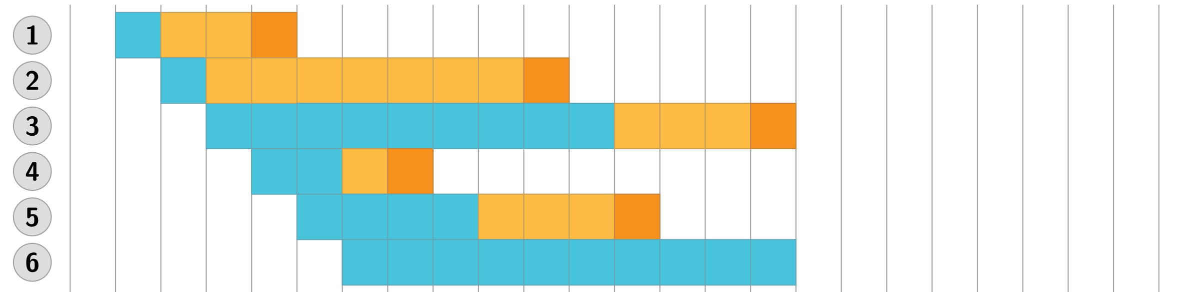
```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```



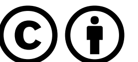
	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											3	...	...	-	-
1	6	-	...	MUL0											

Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

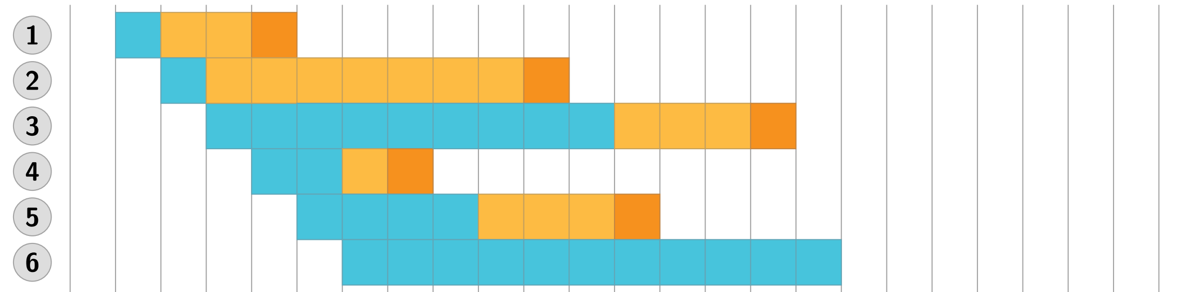


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											3	...	...	-	-
1	6	-	...	MUL0											



Tomasulo: Example

```
1 ld x12, 8(x9)
2 ld x13, 0(x7)
3 mul x17, x13, x12
4 subi x18, x12, 2
5 mul x13, x12, x18
6 add x10, x17, x13
```

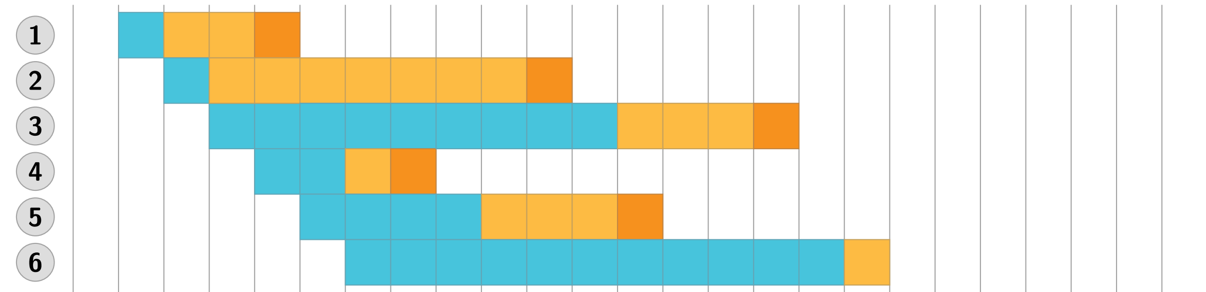


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0											~~3~~	~~...~~	~~...~~	-	-
1	6	...	...	~~MUL0~~											

CC BY

Tomasulo: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

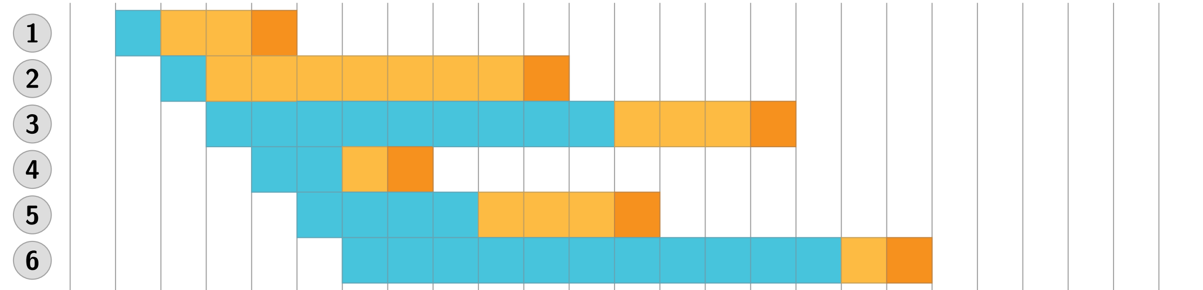


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0															
1	6	…	…												

(c)(i)

Tomasulo: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```

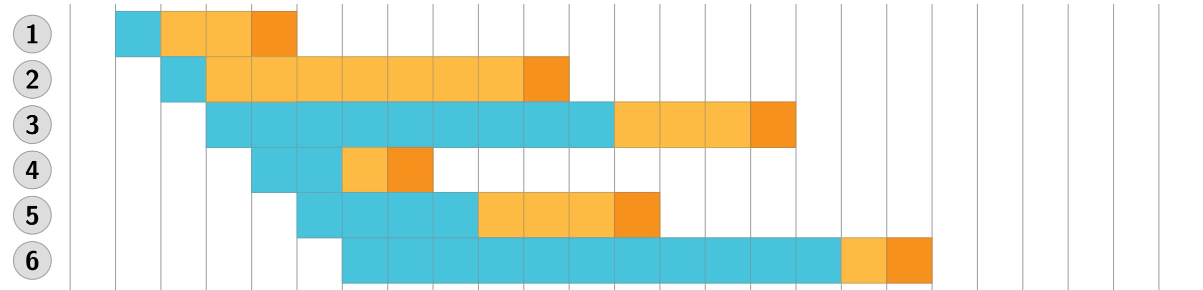


	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0															
1	6	…	…												

©🛈

Tomasulo: Example

```
① ld x12, 8(x9)
② ld x13, 0(x7)
③ mul x17, x13, x12
④ subi x18, x12, 2
⑤ mul x13, x12, x18
⑥ add x10, x17, x13
```



	ALU					LSU					MUL				
	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk	Insn	Vj	Vk	Qj	Qk
0															
1	~~6~~	...	...												

(c)(i)